Computer Science Department

TECHNICAL REPORT

Optimal VLSI circuits for Sorting

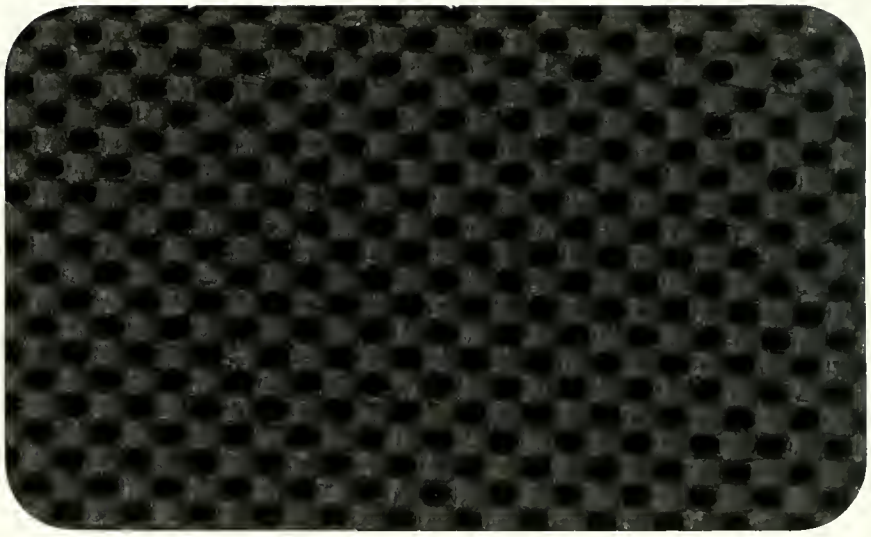
by

Richard Cole†
Alan Siegel‡

Technical Report #172
Ultracomputer Note #86
September, 1985

NEW YORK UNIVERSITY

Department of Computer Science
Courant Institute of Mathematical Sciences
251 MERCER STREET, NEW YORK, N.Y. 10012



Optimal VLSI circuits for Sorting

by

Richard Cole†
Alan Siegel‡

Technical Report #172
Ultracomputer Note #86
September, 1985

New York University
Dept. of Computer Science
Courant Institute of Mathematical Sciences
251 Mercer Street
New York, New York 10012

†This work was supported in part by NSF grant No. DCR-84-01633, and by an IBM faculty development award.
‡This work was supported by Naval Research Laboratory Grant No. N00014-85-M-0260, and by ONR contract number N0014-85-K-0046.

ABSTRACT

This work describes a large number of constructions for sorting N numbers in the range $[0,M]$ for the standard VLSI bit model. Among other results, we attain:

- VLSI sorter constructions that are within a constant factor of optimal size for almost all number ranges M (including $M = N$), and running times T.
- A fundamentally new merging network for sorting numbers in a bit model.
- New organizational approaches for optimal tuning of merging networks and the proper management of data flow.

1. Introduction

We describe VLSI circuits that sort N numbers, in the range 0 to $M-1$; the sorters have area A, and running time $\Theta(T)^{\dagger}$. We have two sorting models: *perimeter sorters*, which have their I/O ports on the boundary of the circuit, and *dense sorters*, which can have I/O ports anywhere in the circuit.

The results divide into three classes, according as $M<N$, $N\le M\le N^2$, and $N^2\le M$. (The partition at N^2 is somewhat arbitrary; it could be at any fixed power of N exceeding 1.) We give sorting circuits for each of these classes, for both the perimeter and the dense models. The bounds are given in the tables below; one column in the tables gives the ratio of our upper bound to the best known lower bound. Our upper bounds match the lower bounds in almost all instances (up to a constant factor). Most of our bounds are AT^2 and AT tradeoffs. However, for very large values of M other tradeoffs also arise. Interestingly enough, for these values of M, the AT^2 and AT tradeoffs apply to two related problems: ranking, and sorting in the case that the inputs can be read twice.

Among other results, we resolve (in the affirmative) the question of tightness for Thompson's original $AT^2=\Omega(N^2)$ lower bound [Thompson, 1979], which applies to essentially planar circuits that sort N numbers in the range $[0,N-1]$. Specifically, we show that an $AT^2 = \Theta(N^2)$ tradeoff holds for $T\in[\log N\log^* N, N^{1/2}]$. For $T=N^{1/2}$, the area is just $\Theta(N)$, the minimum possible area for sorting N numbers in the range 0 to $N-1$ [Ullman, 1984]. In fact, our circuits achieve, for every choice of M and (suitably) maximal T, the minimum area possible for VLSI sorters. Furthermore, these small circuits are all optimal in speed.

The theory of AT^2 bounds for VLSI originates in [Thompson, 1979], where the sorting bound $AT^2 = \Omega(N^2)$, mentioned above, is established. [Thompson, 1980] shows that $AT^2 = \Omega(N^2\log^2 N)$ for a (word-local) restricted class of circuits that sort N numbers in the range $[0,N^2]$, and [Thompson, 1981] demonstrates a circuit construction that satisfies this bound for $T=O(N^{.5})$. In [Leighton, 1984], Thompson's lower bound (for $M = N^7$) is shown to hold without Thompson's restrictions. Moreover, a construction is given that meets this $N^2\log^2 N$ bound for $T \in [\log N, (N\log N)^{1/2}]$. [Bilardi and Preparata, 1984] give an independent (and different) construction that satisfies the same bounds. The Leighton and Bilardi-Preparata sorting circuits apply for any fixed range of size $M = N^{\alpha}$, where α is fixed and greater than 1.

Minimum area bounds for sorters appear for special cases in [Ullman, 1984], and [Leighton, 1984]. The complete solution appears in [Siegel, 1984a] and was independently discovered by [Duris, Sykora, Thompson, and Vrt'o, 1984]. In his thesis, [Bilardi, 1984], shows that another method suffices to give the same results. The AT^2 complexity for sorters,

---

†Note that by using a running time proportional to T, we can describe the limits for T without concern for the fact that the circuit may be a constant factor slower.

in the case $N<M<N^2$, originates in [Siegel, 1984b]; a simpler proof of this result is given in [Siegel, 1985]. In his thesis [Bilardi, 1984] shows that Leighton's method (given in [Leighton, 1984] for $M=N^7$) can be extended to give the same results. The AT^2 bounds can be adapted to apply to other number zones. For $M>>N^2$, the results were independently discovered by [Siegel, 1984c] and [Bilardi and Preparata, 1985b]. For $M<N$, the bounds appear in [Siegel 1984a, as revised for publication, 1985], and are implicit in an independent result of [El Gammal and Pang, 1984].

In independent work, parallel with the results reported in this paper, [Bilardi, 1984] and [Bilardi and Preparata, 1985a] describe several sorting circuits. In particular, they extend the range of optimal sorting circuits (for both dense and perimeter sorters) to include $M \in [N\log^{(i)}N, N^{\alpha}]$, for any fixed i, where $\log^{(i)}N$ is the i*th* iterate of the logarithm; they give non-optimal dense sorter constructions for $M \leq N$ (their results are optimal for constant M). Furthermore, in [Bilardi and Preparata, 1985a] an optimal construction for dense sorters, for $M \geq N^2$, is given, satisfying an $AT/\log A$ tradeoff; a matching lower bound is given in [Bilardi and Preparata, 1985c]. Recently AT lower bounds for dense sorters were shown in [Bilardi and Preparata, 1985b]. Our constructions show that the AT lower bounds are tight. Also, we are able to extend the AT lower bounds to give AT bounds for perimeter sorters, thereby showing our AT upper bounds for perimeter sorters are tight, in most instances.

The first step in our sorting constructions is standard. Both [Leighton, 1984] and [Bilardi and Preparata, 1984] exploit the fact that to sort N numbers, when, say, $M=N^2$, it is useful to use many smaller sorters and then to somehow merge the sorted subsets. The crux of our constructions is how to perform the merging. Leighton achieves the merging by a clever generalization of odd-even merge; as a consequence, he proves that the AT^2 complexity of sorting is not in the sorting subcircuits, but rather in the hardwired permutation networks used to implement columnsort. In contrast, it is provably impossible to built an optimal sorter that utilizes such a permutation network for any number range other than where $AT^2 = \Theta(N^2\log^2 N)$; the data flow would be much too large. Thus our constructions are based on a variety of different approaches.

An essential step is the recognition that we need to use data compression (encoding) to achieve optimal constructions, when $M << N^2$. We are not the first to consider encodings in this context. [Loui, 1983] used encoding to obtain results about the number of bits that must be communicated when sorting (word-local input data) on a ring and on a mesh. His encoding was optimal only for some values of M, which for our purposes is inadequate. In [Siegel, 1984a] an optimal encoding scheme is given (that is optimal to second order terms, in fact). We employ this scheme. Part of the scheme is implicitly presented in [Carter et al, 1978]. In [Bilardi, 1984] and in [Duris et al, 1984] different optimal schemes are proposed, which are optimal to within constant factors. Thus the encoding methods, though important, are not new (and quite possibly predate modern computer science). The key point of the encodings is that a set of X numbers in the range $[0,R]$, $X^2 \geq R \geq X$, can be encoded using $O(X\log 2R/X)$ bits. For $X \geq R$, the encoding requires $O(R\log 2X/R)$ bits. Furthermore, we can merge a pair of encoded sets systolically in unit time per bit. More details are given in appendix A.

Many of our results rely on a fundamentally new sorting construction that we call an *aligned merging network*, described in section 2. All of our sorting results depend on carefully tuned organizational techniques based on the data flow of the sorting circuits. Some constructions exploit an unusual form of pipelining: the running times of the pipeline units are not uniform. Our use of this technique is not straightforward; it requires some care to ensure the optimal bounds are achieved. This technique is used, for example, to achieve the optimal bound for $M=N$, when $T=N^{1/2}$. The sorter has area $\Theta(N)$ which is sufficient to store the encoded input data; the problem is that the unencoded input consists of $N\log N$ bits. The difficulty is that the input has to be sorted before it can be compressed into a representation using just $\Theta(N)$ bits. Our solution inputs just a portion of the input values; the

data is then sorted and compressed, and the process is iterated. However, after some number of iterations, we are obliged to apply further compression to the data that has already been input. We have several recompressions; each successive recompression is carried out less frequently, but takes longer to perform. The hardest part of the construction lies in ensuring that the sum of the areas used for performing the various recompressions is only $O(N)$; this requires careful balancing of running times, areas used, and the frequency of data recompression. This methodology is similar to the funneled pipelining described by [Hochschild, Mayr, Siegel, 1983], and developed further in [Hochschild, 1985]. In those works, however, the technique was applied to graph problems, and had a somewhat different flavor, since the purpose of the technique was to discard data outright, and extra log factors in time and area were of no concern. When aiming for optimal constructions, these factors are important, and for parallel sorting, the entire problem is a matter of removing the log factors.

It is worth noting that our constructions typically use hybrid merging networks. The reason our sorters frequently require several different merging organizations is best understood by examining the lower bound proofs. Typically, a proof follows from demonstrating that a sorting circuit has an information bottleneck. That is, there is a region of the circuit that has just enough perimeter to accommodate the information flow that must, in the worst case, cross the boundary during a specific period of time. When $M<N$, the AT^2 bottleneck for dense sorters occurs for a (minimal) region that inputs $M\log M$ bits (with information content $\Omega(M)$). For $N\leq M\leq N^2$, the bottleneck occurs for a region that inputs $\frac{N}{2}\log M$ bits (with information content $\Omega(N\log 2M/N)$), while for $M>N^2$, it occurs for a region that inputs $N\log N$ bits (with information content $\Omega(N\log N)$). Our circuits are constrained by these bottlenecks; in fact, our constructions typically have three parts. The first uses a merging network that, in general, sorts a set of inputs with a size just below the lower bound bottleneck. The second is a network that performs a merge of data at the bottleneck, and the third is a network that completes the task. Of the three, the third is by far the easiest circuit to design, and typically, the second is the most delicate. (Sometimes the second part divides into two subparts: one, performing a merge of data so that the merged set has a size equal to the bottleneck; the other, performing a merge of sets, each of size equal to the bottleneck.)

In the following sections we describe the various sorting networks. All constructions use, as subunits, optimal sorters of r numbers in $[0,r^2]$, as constructed in [Leighton, 1984]; we call such devices circuit switching sorters, because they can form routing paths that connect each input port with the appropriate sort-determined output port. Alternatively, we could, in all cases but one, use the sorters described in [Bilardi-Preparata, 1984] (which do not compress data either). We will also need several merging networks, and related constructs. Their specific designs require careful detail, but are not technically difficult; as a consequence, their description is left to appendix C. Section 2 introduces aligned merging networks by describing how to build an optimal perimeter sorter for $M=N$ and $T=\log N$. Section 3 describes the construction of an optimal dense sorting circuit for $M=N$ and $T=\log N\log^* N$, and shows how the information flow can be used to organize what is clearly a complicated, delicately tuned cascade of merging networks.

Our other sorting circuits, for $M\leq N^2$, are built using elaborations of the techniques presented in sections 2 and 3, and are described in sections 4, 5 and 6. The sorters for $M\geq N^2$ are by far the simplest to design; they are described in sections 7 and 8. They rely not on data compression, but rather on the distributed sorting of pieces of each number. Were all bits of each output variable to emerge from a local region of circuitry, the requisite data flow would be much too large. This idea of distributing pieces of each number among many cooperating processors is due to [Leighton 1984]. Our contribution in this case is to show how to tune the circuits for optimal results, and how to extend these ideas, when necessary,

via new organizations of circuit cascades.

Remark: It should be noted that our constructions will show how to obtain an encoded form of the inputs, which is implicitly a sorted order. By essentially reversing the methods we describe (but with much less effort), one can obtain, as outputs, the inputs in sorted order. Also, it should be noted that we construct bounded degree circuits that have when- and where-determinate input/output schedules, as defined in [Ullman, 1984].

Dense Sorters: AT^2 Upper Bounds.

size	upper bound	ratio: $\frac{upperbound}{lowerbound}$	comments
$N^2 \le M$ $f \le T \le (N\log N)^{1/2}$, *where* $f = \log\log M + \log N \log 2\log\frac{\log M}{\log N}$	$N^2 \log N \log M$	1	
$N\log N \le M \le N^2$ $\log N \le T \le (N\log\frac{2M}{N})^{1/2}$	$N^2\log^2\frac{2M}{N}$	1	1
$N \le M \le N\log N$ $\log N\,\beta(N, M/N) \le T \le (N\log\frac{2M}{N})^{1/2}$	$N^2\log^2\frac{2M}{N}$	1	2,3
$M = N$ $\log N \le T \le \log N \log^* N$	$N^2\left[\frac{(\log^{(i)}N)^2}{i^2}+1\right]$ *where* $T = i\log N$	$\left[\frac{(\log^{(i)}N)^2}{i^2}+1\right]$	2,3,4
$\log^2 N \le M \le N$ $\max(\log N, \log M \log^* M) \le T \le M^{1/2}$	MN	1	2,3

Remark: We achieve tight AT^2 bounds for all relevant M, and for most of the relevant time interval. Note that A, T, and ratios of upper to lower bounds should all be understood as given up to a constant factor; thus, 1 should be read as, $O(1)$.

Comment 1: The result for $M = N^{1+\alpha}$, $\alpha > 0$, is due to [Bilardi and Preparata 1984] and [Leighton 1984].

Comment 2: $\beta(x,y)$ = least i: $\log^{(i)} x \le y$; $\log^* x = \beta(x,2)$.

Comment 3: For $M < N$ we can write AT^2 bounds for $T \in [\log N, \log M \log^* M]$ similar to the bounds for $M = N$; likewise for $N < M < M\log^{(j)} N$, for any fixed j, and $T \in [\log N, \log N\,\beta(N, M/N)]$.

Comment 4: Setting $i \ge \log^* N - \log^*\log^* N$ makes the ratio equal to 1. This result can be extended to all values of M.

Dense Sorters: AT Upper Bounds.

size	upper bound	ratio: $\frac{upperbound}{lowerbound}$
$N^2 \leq M$ $h \leq T \leq (N\log N)^{1/2}\frac{\log M}{\log N}$ *where* $h = \max[(N\log N)^{1/2}, \log\log M]$	$N\log M\,(N\log N)^{1/2}$ †	1
$N \leq M \leq N^2$	see comment below	
$1 \leq M \leq N$ $\max[\log N, M^{1/2}] \leq T \leq \frac{N}{M^{1/2}\log 2N/M}$	$NM^{1/2}$	1

† This bound applies to ranking circuits for all T as stated above; for sorting cirtuits, it is valid subject to the extra restriction that $T \geq \frac{N\log M}{2^{\Omega((N\log N)^{1/2})}}$. For $\max[(N\log N)^{1/2}, \log\log M] \leq T < \frac{N\log M}{2^{\Omega((N\log N)^{1/2})}}$, (i.e., for $2^{\Omega((N\log N)^{1/2})} < A < N\log M$), the correct tradeoff for sorting circuits is given by $\frac{AT}{\log A} = N\log M$. (This latter tradeoff is due to [Bilardi and Preparata, 1985b].)

Comment: For $N \leq M \leq N^2$, all the circuits, up to the minimum area circuit, are covered by the AT^2 tradeoff.

Perimeter Sorters: AT^2 Upper Bounds.

size	upper bound	$\frac{upperbnd}{lowerbnd}$
$N^2 \le M$ $\log N \log\left(\frac{\log M}{\log N}\right) \le T \le f$, *where* $f = \max[g\,(N\log N)^{1/2}, \frac{g^2(N\log N)^{1/2}}{\log g + \log N}]$ *and* $g = \log\left(\frac{\log M}{\log N}\right)$	$N^2 \log N \log M\, g\left(1 + \frac{g}{\log(2T/g)}\right)^{\ddagger}$	$g^{\dagger}$
$N \le M \le N^2$ $\log N \le T \le (N\log N)^{1/2}$	$N^2 \log N \log \frac{2M}{N}$	1
$N^{1/2} \le M \le N$ $\log N \le T \le (M\log M)^{1/2} \log \frac{2N}{M}$	$NM \log M \log \frac{2N}{M}\left[\frac{\log 2N/M}{\log \frac{2T}{\log 2N/M}} + 1\right]$	$1^{\dagger}$
$1 \le M \le N^{1/2}$ $\log N \le T \le f$, *where* $f = \min[M\log N, (M/\log M)^{1/2}\log^2 N\,]$	$\frac{MN\log^2 N \log M}{\log(2T/\log N)}$	$1^{\dagger}$

† The lower bounds in these cases are for a weaker model.

‡ This bound applies to ranking circuits for T indicated above, and to sorters for times $g\log N \le T \le gN^{1/2}\max[(\frac{\log N}{g})^{1/2}, 1]$. For sorting circuits and larger times (i.e., for $N\log N < A < N\log M$): $AT = O(\max[N^{3/2}\log M \log \frac{2A}{N\log N}, N\log M\,(N\log N)^{1/2}\log^{1/2}\frac{2A}{N\log N}])$.

Remark 1: Note that A, T, and ratios of upper to lower bounds should all be understood as given up to a constant factor; thus, 1 should be read as $O(1)$.

Remark 2: For $M \ge N^2$ we believe the lower bound is weak. However, we can build a 'verifier' circuit that achieves this lower bound. That is, we input the rank of each item as well as its value, in a where- and when-determinate format, and the circuit verifies these ranks. This suggests it may be hard to improve the lower bound.

Remark 3: The result for $M = N^{1+\alpha}$, $\alpha > 0$, is due to [Bilardi and Preparata 1984] and [Leighton 1984].

Perimeter Sorters: AT Upper Bounds.

size	upper bound	$\frac{upperbnd}{lowerbnd}$
$N^2 \le M$ $f \le T \le (N\log N)^{1/2}\frac{\log M}{\log N}$, *where* $f = \max[g\,(N\log N)^{1/2}, \frac{g^2(N\log N)^{1/2}}{\log g + \log N}]$, and $g = \log\left(\frac{\log M}{\log N}\right)$	$N\log M\,(N\log N)^{1/2}$ ‡	1†
$N \le M \le N^2$	see comment below	
$N^{1/2} \le M \le N$ $(M\log M)^{1/2}\log\frac{2N}{M} \le T \le \frac{N(\log M)^{1/2}}{M^{1/2}\log\frac{2N}{M}}$	$N\,(M\log M)^{1/2}$	1
$1 \le M \le N^{1/2}$ $f \le T \le \frac{N(\log M)^{1/2}}{M^{1/2}\log N}$, *where* $f = \min[M\log N, (\frac{M}{\log M})^{1/2}\log^2 N\ ]$	$N\,(M\log M)^{1/2} + \frac{N\log M\log N}{\log(2T/\log N)}$	1†

† The lower bounds in these cases are for a weaker model.

‡ These bounds apply to ranking circuits, not sorting circuits. For bounds on sorting circuits, see the table of AT^2 results for perimeter sorters.

Comment: For $N \le M \le N^2$, all sorting circuits, up to the minimum area circuit, are covered by the AT^2 tradeoff.

2. Aligned Merging

We illustrate the aligned merge by describing the perimeter sorter for $M = N$, $T = \log N$.

The basic strategy is to group the input numbers into $\log N$ sets of $N/\log N$ inputs each, called *portions*. The part of the circuit that handles a portion of inputs is called a *block*. We number the blocks 1 through $\log N$. The blocks are laid out in a vertical column. Each block includes a circuit switching sorter running in time $O(\log N)$. The outputs of these sorters are merged, using an *aligned merge*, described below. Each block is of size $O(N/\log N) \times O(N/\log N)$, so the perimeter sorter has size $O(N) \times O(N/\log N)$.

To perform the merge we connect the blocks using $O(N/\log N)$ merging channels, each of constant width. Each channel has one connection to each block. For each channel, for each block, we wish to feed a packet (containing a small set of numbers), and to merge these $\log N$ packets in $O(\log N)$ time, at which point the sort should be complete. To enable this, we impose a *uniformity criterion*: each packet can contain at most $\log N$ numbers drawn from a small range of values ($O(\log N)$). So the first task is to divide the (locally sorted) inputs in each block into packets. We call the values separating the packets *breakpoints*.

The breakpoints are chosen to be the following $2N/\log N$ values (some of which may be repeated): every $\log N$*th* item in the sorted order in each block, and the values $i \log N$, $0 \le i \le (N-1)/\log N$. The breakpoints are broadcast to all the blocks, which can easily be done in $O(\log N)$ time using the $O(N/\log N)$ channels. With each breakpoint we record two numbers, the *block number* and the *position*. The block number is the number of the block in which the breakpoint originated, or zero for the $N/\log N$ breakpoints given the values $i \log N$, $1 \le i \le N/\log N$. The position is the rank of the breakpoint in the set of sorted items in the block from which the breakpoint originated. It is also convenient to associate with each input its block number and position.

In each block, we sort the inputs and breakpoints. (The sort key for input and breakpoint data is the triple <value, block number, position> which is ordered lexicographically.) For each pair of adjacent breakpoints we encode the items between them. Since two adjacent breakpoints are at most $\log N$ apart in value, and since there are at most $\log N$ items between them, this encoding uses $O(\log N)$ bits. We term the encoded items a packet. The packet is associated with the breakpoint preceding the items. (It is easy to form the packets in $O(\log N)$ time.) It is important to use the coding that is optimal for the final sorted order (i.e. $O(\log N)$ items in an $O(\log N)$ range). This precoding allows us to merge packets with the same header both quickly and on the fly. We remark that such an encoding is, in general, not optimal for the outputs of individual blocks.

Next, we *align* packets with equal headers. That is, in each block we sort the packets according to the breakpoint value (using a circuit switching sorter, running in time $O(\log N)$). The packet formation and alignment is essentially a routing problem. We solve it by creating $2N/\log N$ dummy packets with breakpoints as headers, and sorting the input numbers and dummies, thus interlacing the data and packet headers. The data is then transferred to the packet headers in an encoded format. Alignment occurs after another lexicographical sort, where packets are counted larger than input numbers. Packets with the same breakpoint are thus aligned in the same position in the different blocks.

There is still one detail preventing us from merging the aligned packets: a distributed set S of aligned packets may contain up to $\log^2 N$ items ($\log N$ items from each of $\log N$ blocks). Since it is easy to compute how many items are in S as well as the partial sums, (if we record with each packet its size), we can allocate an appropriate number of channels for merging the items in S, and assign local packets a channel number so that $O(\log N)$ numbers will be merged on any channel. (For each packet P we need to compute two numbers: the *global displacement*, the number of channels allocated to packets from sets preceding S, and the *local displacement*, the number of channels allocated to packets in S preceding P; a packet in S precedes $P \subset S$ if it is in a block with a smaller block number and it is not allocated the

same channel as P. Details are left to the reader.) We generate a corrected set of dummy packet headers in each block, and perform a second alignment of the data (with "just" two more circuit switching sortings) so that our uniformity criterion is now globally satisfied. (The sorts proceed as follows. In each block we create a dummy header for each channel; we sort the dummy headers together with the packets, the key being the channel number. Then we copy each packet to the dummy having the same channel number; this dummy is adjacent to the packet following the sort. We resort the data lexicographically, packets being counted larger than dummies. Now each dummy is aligned at the correct channel, so the packets are correctly aligned.)

Next, we merge the items in the packets allocated to one channel, on the fly, in time $O(\log N)$, obtaining a sorted packet, also of length $O(\log N)$. Each of these sorted packets is placed, essentially, in the block from which it should eventually be output.

Actually, we may not be able to carry out this last step because the $|S|$ items associated with the same breakpoint have not been completely merged, in general. However, there are only two (adjacent) blocks to which each item may belong, since $|S| \leq \log^2 N$, so each packet is routed to one of the two blocks to which its items belong (if there is any doubt). We also ensure that exactly $N/\log N$ items are placed in each block. (This may require splitting the output of some channels into two packets.) The packets are decoded, and the items in each block are sorted, once more. It turns out that every item is within $\log^2 N$ places of its correct sorted position and thus it is easy to complete the sort.

In summary, the essence of the method is as follows. Divide the inputs into portions and sort each portion. Obtain a global sample of the inputs, called breakpoints, which partition the sorted inputs relatively evenly. Using the breakpoints, partition each portion of inputs into packets. *Align* packets that cover the same range and merge the items they contain. To carry out the merge efficiently, it is vital that the numbers in each packet be coded using the method that is most compact for the output.

The merging channel is trivial to construct, but nevertheless should be sketched in more detail. Partial sums are easily computed on a binary tree, and our channel is in fact, an instance of a tree. Merges are done systolically, and can also be done (using distributed systolic queues) on a tree. In the case $N = M$, and $T = \log N$, a channel is simply a depth $\log N$ tree with $\log N$ leaves, which is essentially a simple chain! This is why the channel has width $O(1)$. In general, we must extrapolate this simple tree structure to accommodate additional leaves (of depth $O(T)$) and account for their increased layout width. The nature of such trees has been analyzed in [Yao, 1981]. In any case, our merging paths and merge packets are always of length $O(T)$. The layout height of such trees, though, will in general add to the width of the merging circuit and hence the area of the sorter.

It may be appropriate to comment on the sorting method and the encoding scheme. If we consider a horizontal cut of the network we see that $O(N)$ bits cross it, as is required for an optimal sorter. In the vertical direction however, we have a surprise: $O(N \log N)$ bits cross a vertical cut. Our aligned merge uses suboptimal encodings to attain an optimal sort! We see this method is suitable only for a non-square network. Optimal perimeter sorters, however, must have a skewed aspect ratio, so there is no inconsistency in our construction. We remark that a minimum area dense sorting circuit is square, and thus this method is not by itself adequate for designing optimal dense sorters.

We extend the construction to other values of T, $\log N \leq T \leq N^{1/2}$. The construction is essentially the same. In particular, the number of blocks remains at $\log N$, but the running time of each phase is increased to $O(T)$. The packet size is also increased to $O(T)$, while both the number of channels and the number of breakpoints is reduced to $O(N/T)$. The area used by each circuit switching sorter is reduced to $O(N^2/T^2)$. It is easy to see that our construction requires no substantial changes. We deduce

Lemma 1: For $M = N$, and $\log N \le T \le N^{1/2}$, there are perimeter sorters with $AT^2 = \Theta(N^2 \log N)$.

Extending this construction to larger values of M, $N \le M \le N^2$, is straightforward.

Lemma 2: For $N \le M \le N^2$, and $\log N \le T \le (N \log 2M/N)^{1/2}$, there are perimeter sorters with $AT^2 = \Theta(N^2 \log N \log 2M/N)$.

Proof: The construction is much as above, with a few parameters changed. In particular, we use $\frac{\log N}{\log 2M/N}$ blocks. The only part that needs a little care is the systolic merging (in the aligned merge). □

We complete our study of AT^2 tradeoffs for perimeter sorters with $N \le M \le N^2$ in section 4.

3. Dense Sorters

Recall that [Thompson, 1979] establishes the bound $AT^2 = \Omega(N^2)$ for dense sorters of N numbers in the range $[0,N]$. We now show that this bound is tight.

Loosely speaking, the dilemma in achieving an optimal sorting circuit is as follows. If a circuit is to sort optimally for $M = N$, then the total flow of information across a line partitioning the circuit into halves must be $O(N)$ bits [Thompson, 1979]. Moreover, the $O(N)$ bits must, essentially, describe all the numbers that comprise the input data. This difficulty, on first thought, might not seem too serious, since N numbers in the range $[0,N]$ have an information content of about $2N$. However, before N numbers can be compressed into such a minimum representation, they have to be sorted (or the equivalent). So there is a paradoxical obstacle that must be overcome: to sort the numbers they must be compressed; to compress them they must be sorted. Furthermore, there are two difficulties intrinsic to our divide and conquer approach. First, each level presumably requires $\log N$ time. Second, the sum of the information content of the parts is critically larger than the information content of the whole. As we shall see, this suggests a $\Theta(\log N)$ time, $AT^2 = N^2$ sorter is unattainable.

The first step in our strategy is to group the inputs into $\log^2 N$ sets of $N/\log^2 N$ inputs each, called portions. Again, each portion of inputs is handled by a block of the circuit. These blocks are laid out as a $\log N \times \log N$ array. Each block includes a circuit switching sorter running in time $\Theta(T)$. The outputs of these sorters are then merged, as described below. Each sorter takes area $O(\frac{N^2}{T^2 \log^2 N})$, so the total area used by these sorters is $O(N^2/T^2)$. Thus, if we could merge the output of $\log^2 N$ blocks in $\log N$ time, and use $N^2/\log^2 N$ area for the merging network, we would have $AT^2 = O(N^2)$. We do not, however, know how to accomplish such an optimal $\log N$ time merge; in fact, we conjecture that no such circuit exists. Instead, we use somewhat smaller and slower blocks to reduce the total area so that $AT^2 = O(N^2)$ for $T = \log N \log^* N$.

The choice of $\Theta(\log^* N \log N)$ time is motivated by the necessities of data compression at various stages of the sort. To be specific, let us imagine we are aiming for an $O(\log N)$ sorting time. In $\log N$ time we can transmit up to $N/\log N$ bits across the boundary of a block. In this time we wish to transmit (in encoded form) all $N/\log^2 N$ numbers input into this block. A little thought shows that any encoding of these $N/\log^2 N$ numbers needs at least $\log\log N$ bits per number, on the average. Suppose we merge sets of outputs from the blocks, without encoding them any more densely. Once we have combined the outputs of $(\log N/\log\log N)^2$ of these blocks (arranged in a square shape), $O(\log N)$ time will be required to transmit the merged ensemble of bits across the boundary of this square. If we try to do any further merging without recoding, $O(\log N)$ time will be insufficient for transmitting the merged set of bits across the boundary of the square merging region. (Notice that a region with a skewed aspect ratio (and hence larger perimeter) will not improve the data throughput, since

the internal merge of data will be limited by the shorter dimension.) Thus it seems we are obliged to recode the inputs at this point. The merging region has data representing $N/(\log\log N)^2$ inputs, and a perimeter of length $N/\log N \log\log N$. By iterating the argument, we deduce there must be at least $\log^* N$ recoding stages in the sorter. These recoding levels form the basis of our construction. This contrasts with the perimeter sorter where we are able to have just one recoding. (The above discussion is intended to motivate our constructions; it is not likely to provide a lower bound proof, since recoding can be done on the fly in constant time. Our discussion also illustrates some, though not all of the difficulties that must be overcome in designing an optimal sorter for this number range. We believe that the true obstacle to an optimal $\log N$-time sorter is the data rearrangement inherent to merging.)

There a few more properties that can be anticipated about our next construction. If, for $M = N$, there is an AT^2-optimal sorter based on merging, then the final merging network must run in time $\Theta(T)$, since it must transmit $\theta(N)$ bits across a cut of length $\Theta(N/T)$. On the other hand, if we have $\log^* N$ levels of merging networks, and if they cannot be pipelined, then at least half of them must run in time $O(\log N)$ to achieve a sort in $T = \log N \log^* N$. Thus we adopt a two-type merging structure; the first type will comprise $\log^* N$ levels, each running in $\log N$ time, and the second type will contain a hierarchy with running times interpolating between $\log N$ and $\log N \log^* N$. Moreover, the final merging network will consume area $\Theta(N^2/T^2)$; but each level of the type 1 merging networks ought to consume area $O(N^2/T^2 \log^* N)$ (or better still, have side length $O(\frac{N}{T \log^* N})$) to remain within the area bound. Since each level of our type one merging network is $\log^* N$ faster and smaller than our complete sorter, the parameters stating just how many inputs can be absorbed per level (i.e. without recoding) have to be recomputed. There will still be only $O(\log^* N)$ levels, however. Achieving all these features simultaneously is the major challenge of the construction.

Our sorter turns out to use $\log^* N$ levels of a merging network that, at level l, merge the outputs from about $\left[\frac{\log^{(l-1)} N}{\log^{(l)} N (\log^* N)^2}\right]^2$ level $l-1$ networks. Each level runs in time $O(\log N)$. This method breaks down at a level comprising approximately $(\log^* N)^8$ networks. The merging is completed with, effectively, an H-tree of binary merging devices, whose $\log\log^* N$ levels have running times that form a geometric series. Careful tuning is needed to keep the area within bounds. The details can be found in appendix B. We can now deduce

Lemma 3: For $M = N$, $T = \log N \log^* N$, there exist VLSI sorters with $AT^2 = \Theta(N^2)$.

This result can be extended to larger values of T in many ways. For, say, $T \leq \log^{5/4} N$, we can simply proceed as before, but rescale the number of bits per wire up by a $T/\log N \log^* N$ factor, and rescale the wire count down correspondingly. For $\log^{5/4} N \leq T \leq (N/\log N)^{1/2}$, it is simpler to use an H-tree exclusively. There are $\frac{1}{4}\log\log N$ stages, where stage k has $2^k T/\log^{1/4} N$ bits per wire. A stage comprises two levels of a binary tree, so that the merges are all pairwise. With this organization, the task of going from a nearsort to a perfect sort is quite simple, even for large T. We choose the blocks to take $N/\log^2 N$ inputs (they are essentially circuit switching sorters, except that they produce their output in encoded form). It is clear this yields $AT^2 = \Theta(N^2)$ for $\log N \leq T \leq (N/\log N)^{1/2}$. Since $T = (N/\log N)^{1/2}$ is the slowest the circuit switching sorters can run, for $N/\log^2 N$ inputs, a different sorting strategy is required for larger T.

Our construction, for $T = N^{1/2}$, uses the following merging network (type 3), indexed by the parameter x. The network is a box configured around a 4×4 array of units. Each unit produces, as output, the encoded form of $N/(64x^3)$ inputs. The merging network accumulates 4 waves of outputs from these units, and produces these N/x^3 input values in encoded form as

its output. The outputs from the merging network leave on $N^{1/2}\log x/x^{3/2}$ wires. The increase in side length due to the merging network is $O(N^{1/2}\log x/x^{3/2})$, so an $x \times x$ array of networks incurs a side length increase of $O(N^{1/2}\log x/x^{1/2})$. The running time of the merging network is the maximum of $O(N^{1/2}/x^{3/2})$, and the time for the four waves of inputs to be output by the units. (We remark that the latter term turns out to be the larger.) The merging network is described in appendix C.

We build $1/2\log\log N$ levels of the type 3 merging networks. The units at the bottom level of our construction are essentially circuit switching sorters that take $N/\log^3 N$ inputs (except that the output appears in encoded form on $N^{1/2}\log\log N/\log^{3/2} N$ wires). Thus, there are $\log^2 N$ circuit switching sorters at the bottom level that process $\log N$ waves of $N/\log^3 N$ inputs. At level $i+1$, the units are the boxes at level i. Our construction uses area $O(N)$ (the increases in side length due to the levels of merging networks form a modified geometric series).

The sorters at the bottom level run in time $O(N^{1/2}/\log N)$ time per wave. A merging network k levels up requires 4^k waves of inputs to the sorters, and thus runs in time $O(4^k N^{1/2}/\log N)$. We observe that as a merging network is outputing one wave it can be processing the next wave. Thus, the running time of the whole system is $O(N^{1/2})$. We deduce

Lemma 4: For $M = N$, and $T = N^{1/2}$ there exist VLSI sorters with $AT^2 = \Theta(N^2)$.

It is fairly straightforward to construct circuits for the remaining values of T, $(N/\log N)^{1/2} \le T \le N^{1/2}$. We use the strategy just described, with two changes. The $\log^2 N$ units at the bottom level of the construction are circuit switching sorters which take $N^2/(T^2\log^3 N)$ inputs per wave, and run in time $\Theta(N/(T\log N))$ per wave. We switch to an H-tree after $1/2\log\left(\frac{T^2\log N}{N}\right)$ levels of the type 3 merging networks, that is, when all of the inputs have been read by the circuit. (We have to choose type 3 mergers of an appropriate size and speed. Since the idea is exactly the same as in the special case presented above, the details are left to the reader.) We obtain

Lemma 5: For $M = N$, and $\log N \log^* N \le T \le N^{1/2}$, there exist VLSI sorters with $AT^2 = \Theta(N^2)$.

It is not difficult to extend these results to number ranges where $N \le M \le N^2$. For the time range $\log N \log^* N \le T \le N^{1/2}$, we could, for example, simply replace each wire by a bus of $\log 2M/N$ wires. For very slow times such as $T = (N\log 2M/N)^{1/2}$, it suffices to increase the wire (and packet) count for each stage by a factor of $\log^{1/2} 2M/N$. All packet lengths (and merging times) are also increased by the same factor. It is convenient (though not necessary) to increase the number of inputs (per wave) to each circuit switching sorter by a factor of $\log^3 2M/N$ and to decrease the number of sorters by $\log^2 2M/N$. The numbers of input waves and merging stages are adjusted accordingly. (We remark that the circuit area when $T = (N\log 2M/N)^{1/2}$, is proportional to the information content of the input, and no further increase in time, therefore, is useful [Siegel 1985].) For the very fast times when $\log N\, \beta(N, 2M/N) \le T < \log N \log^* N$, the construction is analogous to the case $M = N$ for small T: the principal modifications are that the number of circuit switching sorters is reduced by a factor of $\log^2 2M/N$ (with a corresponding increase in the number of inputs), that $\log^* N$ is replaced by $\beta(N, 2M/N)$ in the parameterized constructions, and that the wire counts (and numbers of packets) are adjusted to accommodate the increased information flow. Here $\beta(x,y) = \min i : \log^{(i)} x \le y$. As a consequence, we have,

Lemma 6: For $N \le M \le N^2$, and $\log N\, \beta(N, 2M/N) \le T \le (N\log 2M/N)^{1/2}$, there exist VLSI sorters with $AT^2 = \Theta(N^2 \log^2 2M/N)$.

4. Perimeter Sorters: $N \le M \le N^2$

We complete the study of AT^2 tradeoffs for $M \in [N, N^2]$ by constructing optimal perimeter sorters for large times, up to $T = (N\log N)^{1/2}$, at which point we obtain minimum

area circuits. We start with the case $M = N$. Since constructions for $T \leq N^{1/2}$ are given in section 2, it suffices to suppose $T = kN^{1/2}$, where $1 \leq k \leq (\log N)^{1/2}$. The circuit uses $\log N$ blocks, configured as a vertical column. Each block comprises, essentially, a circuit switching sorter that sorts k^2 waves of $N/(k^2 \log N)$ inputs, at the rate of one wave per $O(N^{1/2}/k)$ timesteps. We use aligned merging networks to combine, for each wave, the encoded outputs from $\log N/k^2$ consecutive blocks. Thus there are k^2 alignment *structures*, each of which has N/k^4 inputs per wave. The structures run in time $O(N^{1/2}/k)$ per wave of inputs, have width $O(N^{1/2}/k)$, and length $O(N^{1/2} \log N/k^3)$. Altogether, they have total length $O(N^{1/2} \log N/k)$ and width $O(N^{1/2}/k)$. We complete the sort with a $(2 \log\log k)$-level tree of type 6 merging networks (described in appendix C).

The type 6 merging networks are interconnected in a sideways binary tree organization that has one network unit per node. Nodes belonging to the same level are located along vertical columns. A merging wave, for a network unit, uses the outputs from two waves of its two children at the next lower level. Thus the packet length and running time per wave double at successive levels of the tree. The wire count is adjusted accordingly. For our current application (where $M > N$) a l-th level merging network has width proportional to its $(N^{1/2} 2^l (4 \log k - 2l)/k^3)$-wire bus, and has length $O(N^{1/2} 2^l \log N/k^3)$. Its running time and packet length are $O(N^{1/2} 2^l/k)$.

The type 6 merging networks at the bottom level read the outputs from the structures which encode N/k^4 inputs per wave, described above. At this bottom level, strings of $O(k^2/\log k)$ consecutive packets from each structure are merged (i.e. concatenated) together. No sorting is needed, since the aligned merge gives a perfect sort for the contents of each structure. The widths of the merging networks form a modified geometric series, the last term and the total being $O(N^{1/2}/k)$. The running times of the merging networks can be overlapped (that is, while the mergers at one stage are working on the k*th* wave of inputs, the mergers at the next stage down are working on the $k+1$*st* stage). Thus the total running time for the networks is $O(kN^{1/2})$. We obtain

Lemma 7: For $M = N$, and $\log N \leq T \leq (N \log N)^{1/2}$, there exist VLSI perimeter sorters with $AT^2 = \Theta(N^2 \log N)$.

Proof: The structures process all the waves of input in time $O(kN^{1/2}) = O(T)$. The structures together have length $O((N^{1/2} \log N)/k)$, which is therefore the length of the sorter. The structures have width $O(N^{1/2}/k)$, as do the type 6 mergers. Hence $AT^2 = \Theta(N^2 \log N)$. □

We extend the result to $N \leq M \leq N^2$. The idea is to replace $\log N$ by $\log N/\log(2M/N)$. We have obtained the AT^2 tradeoff for $T \leq (N \log(2M/N))^{1/2}$. So let $T = k(N \log(2M/N))^{1/2}$, $1 \leq k \leq (\log N/\log(2M/N))^{1/2}$. We use blocks as before: each one takes $N/(k^2 \log N/\log(2M/N))$ inputs. We merge the inputs to $\log N/(k^2 \log(2M/N))$ blocks, using an aligned merge. This yields structures that have N/k^4 inputs. We use k^2 of these structures, receiving their inputs in k^2 waves. As above, we combine the waves of inputs to these structures using the type 6 merging networks. We obtain

Lemma 8: For $N \leq M \leq N^2$, and $\log N \leq T \leq (N \log N)^{1/2}$, there exist VLSI perimeter sorters with $AT^2 = \Theta(N^2 \log N \log(2M/N))$.

5. Dense Sorters: $1 \leq M \leq N$

We start by obtaining an AT^2 tradeoff. We will use type 4 merging networks, parameterized by x. This network is a box configured around an 8×8 array of units, each of which uses $O(M \log(2x/M))$ bits to output $x \geq M$ values in encoded form. The merging network produces as output its $64x$ inputs in encoded form, using $O(M \log(128x/M))$ bits. Let $p = 1/6 \log(2x/M)$. A level p merging network produces its output on $\frac{M}{T} 4^{p+1}$ wires. The increase in side length due to one instance of a p-th level merging network is $O(M \cdot 4^{p+1}/T)$. We note that for $x \geq 64M$, the output of our merging network has (a maximum) information

content bounded by twice that of a single input unit. Evidently the wire count of a p-th level merging network is 4 times that of a $(p-1)$-th stage, so as long as a packet header needs only a nominal portion of the packet length for an output wire, this merging network has the property that an output packet has a bit length at most half the length of a unit's packet. Also, the wire count will never exceed x. Consequently, our network can merge in time $O(\max[T\cdot(1/2)^{p+1},\log x])$. (As a matter of courtesy to the reader, we include both terms when computing the total merging time, if it is not immediately clear which term is dominant.) We show how to construct these merging networks (type 4) in appendix C.

Case 1: $\log^2 M \le T \le M^{1/2}$.

Case 1a: $N \le M^2$. We build a hierarchy of the type 4 merging networks, starting with an $(N/M)^{1/2}\times(N/M)^{1/2}$ array of units, each of which is a dense sorter for M numbers and runs in time $O(T)$. The area used by these units is $O(N/M\cdot M^2/T^2)$. The hierarchy uses $1/6\log(2N/M)$ levels of type 4 merging networks. The time taken to sort is $O(T+T/2+\ldots)+O(\log N\log(2N/M)) = O(T)$. The additional side length needed for the merging networks forms a geometric series of the form $O((NM/T^2)^{1/2}+1/2(NM/T^2)^{1/2}+\ldots)$. So the total area used is $O(NM/T^2)$.

Case 1b: $M^2 \le N$ and $\max[\log^2 M,\log N]\le T\le M^{1/2}$. We start with an $N^{1/2}/M\times N^{1/2}/M$ array of units, each of of which is an M^2 input, T-time optimal dense sorter, as constructed in case 1a. Each such unit is an $M^{3/2}/T\times M^{3/2}/T$ square and has $M^{3/2}/T\ge M$ output wires. Our encoding scheme, for the output of each unit of M^2 input numbers, is modified to produce M counts, each of $2\log M$ bits: the count on wire j is just the number of instances of the number j among the unit's inputs. The units are interconnected in an (M-multiple) H-tree of N/M^2 unit-leaves. Each internal tree node is a systolic ripple adder. The additional area used by the H-tree is $O(N)$. The time used by the H-tree is $O(\log N)$, giving a total sorting time of $O(\log N+T)$. We obtain

Lemma 9: For $\max[\log^2 M,\log N]\le T\le M^{1/2}$, and $M\le N$, there exist VLSI sorters with $AT^2=\Theta(MN)$.

Case 2: $\max[\log N,\log M\log^* M]\le T\le\log^2 M$. We have to proceed more delicately, as was the case for $M=N$ and $T=\log N\log^* N$. Note that M cannot be too small: $M\ge 2^{(\log N)^{1/2}}$. The construction begins as in case 1a, with N/M units, each of which is an optimal T-time dense sorter of M numbers. We then use $\min[\log_{64}N/M,\ \log\log^* M]$ levels of the type 4 merging network. The sorting will be finished if $N/M\le(\log^* M)^6$. Otherwise, each merging network, at the top level, will have $M(\log^* M)^6$ inputs, and output on $M(\log^* M)^2/T$ wires, with at most $O(T/\log^* M)$ bits per wire. (For $T=\log M\log^* M$ this is $O(\log M)$ bits per wire). Then another network hierarchy, type 5, is used for further merging, and finally, an H-tree is used, if necessary, to complete the sort as in case 1b above.

The type 5 merging network is parameterized by x where $x\ge(\log^* M)^3$, and $2^{x/(\log^* M)^2}\le T\log^* M$. It is a box configured around an array of $(2^{x/(\log^* M)^2}/x)\times(2^{x/(\log^* M)^2}/x)$ units that each inputs data representing Mx^2 values, and that outputs these numbers in encoded form (using $O(M\log x)$ bits). The outputs from each row of units in the array is align merged (as described in appendix C) and then the row outputs are merged (by packet sorting rather than alignment). The resulting merge outputs its $M4^{x/(\log^* M)^2}$ inputs in encoded form using $O(Mx/(\log^* M)^2)$ bits. It runs in time $\max[T/\log^* M,2^{x/(\log^* M)^2}/x] = T/\log^* M$, and has $Mx/(T\log^* M)$ output wires. The increase in side length due to the row merging network is $O\left(\frac{M2^{x/(\log^* M)^2}}{T\log^* M}\right)$. Since there are $O(N/M4^{x/(\log^* M)^2})$ merging arrays of the networks at this level, the total side length increase is $O(N/M4^{x/(\log^* M)^2})^{1/2}\left(\frac{M2^{x/(\log^* M)^2}}{T\log^* M}\right) = O(\frac{(MN)^{1/2}}{T\log^* M})$. The merge of the rows contributes the same increase in side length. We show how to construct this merging network (type 5) in appendix C.

We use up to $\log^*M - \log^*\log^*M$ levels of the type 5 merging network so that at the top level the number of inputs to a merging network is between $\min[N, M(T\log^*M)^2]$ and $\min[N, M(\log(T\log^*M)\cdot(\log^*M)^2)^2]$. These networks induce a total side length increase of $O((MN/T^2)^{1/2})$, and the total running time for the hierarchy is $O(T)$.

It is possible, however, that further merging is still required. By adding one extra stage of a (possibly subsized) type 5 merging network, if necessary, we may suppose that at the top level of our hierarchy, $2^{x/(\log^*M)^2} = T\log^*M$. In this case, each top level network has a side length of at least $\frac{M2^{x/(\log^*M)^2}}{T\log^*M} \geq M$. Then the merge is completed by switching to the encoding described in case 1b, and by using an M-multiple H-tree. We obtain

Lemma 10: For $\max[\log N, \log M\log^*M] \leq T \leq M^{1/2}$, and $\log^2 N \leq M \leq N$, there exist VLSI sorters with $AT^2 = \Theta(MN)$.

Next, we obtain an AT tradeoff for larger values of T. First, we describe the construction for $\max[\log N, M^{1/2}] \leq T \leq \min[N/M^{3/2}, N/(M^{1/2}\log\frac{2N}{M})]$. The upper bound on T has a natural interpretation. When $T = N/M^{3/2}$, the inputs are pipelined into a single AT^2-optimal sorter and the output is processed further. For faster times, we use several such sorters in parallel. The $N/(M^{1/2}\log\frac{2N}{M})$ time bound occurs for the smallest circuit possible.

Case 1: $\max[\log N, M^{1/2}] \leq T \leq \min[N/M^{3/2}, N/(M^{1/2}\log\frac{2N}{M})]$. The first stage of our constructions uses optimal sorters (described in Lemma 10) to sort sets of M^2 numbers in time $O(M^{1/2})$, and area M^2. We use $N/(TM^{3/2}) \geq 1$ of these sorters. We refer to the inputs that arrive in $O(M^{1/2})$ time as a *wave* of inputs. There are $T/M^{1/2}$ waves of inputs. The sorters are connected by an M-multiple H-tree. The area for such an $N/(TM^{3/2})$-leaf tree is $NM^{1/2}/T$. Each H-tree is used to count the number of occurrences of one input value. An internal node comprises a single bit full adder plus a few bits of storage and control. This construction clearly has $AT = O(NM^{1/2})$. Now, we need $T \geq \log N$ as always, and $T \geq M^{1/2}$ which is proportional to the sorting time of our basic sorter. In addition, we need $\Theta(M\log N)$ space at the top level of the tree for the counts. Actually, we need only guarantee $\Theta(M\log 2N/M)$ space, since $M < N^{2/3}$. Sufficient area, in this case, is implicit in the $AT = \Theta(NM^{1/2})$ tradeoff, since $T \leq N/(M^{1/2}\log\frac{2N}{M})$.

Case 2: $\max[M^{1/2}, N/M^{3/2}] \leq T \leq N/(M^{1/2}\log(2N/M))$. Let $T = N/xM^{1/2}\log(2N/M)$, and $A = xM\log(2N/M)$. At the lowest level, we use one AT^2-optimal sorter described in Lemma 10. It has area A, and sorts A numbers. It runs in time $\Theta(M^{1/2})$. There are $N/xM\log\frac{2N}{M}$ waves of inputs. The sorter outputs its data, for each wave, in a slightly suboptimal encoding. The optimal encoding would represent, of course, A numbers in the range $[0, M-1]$. Instead, we use the encoding for $A\cdot\log\frac{2N}{M}$ numbers. The total number of bits used is still $O(M\log(x\log\frac{2N}{M}))$. The output is sent in $O(M^{1/2})$ length packets on $A^{1/2}$ wires to a merging sorter that stores its output in this same format and that sequentially merges $\log\frac{2N}{M}$ waves with its previous (and initially empty) output. (Of course, if there were fewer waves of inputs, the sort would be completed when all the inputs have been merged.) Each wave is merged with the previous output in $\Theta(M^{1/2})$ time. This merging sorter also has area A. It outputs the merge of $\log 2N/M$ waves to a unit that recodes the data into the format used to represent N numbers in the range $[0, M-1]$ (the recoding method is discussed in appendix D). Now, the packets are of length $(M\log 2N/M)^{1/2}$. The $M\log 2N/M$ bits are sent to a similar (and final) merging sorter that merges these group waves at the rate of one merger per $M^{1/2}\log\frac{2N}{M}$ time. This sorter has area A. We deduce

Lemma 11: For $1 \le M \le N$, and $\max[\log N, M^{1/2}] \le T \le N/(M^{1/2}\log\frac{2N}{M})$, there exist VLSI sorters with $AT = \Theta(NM^{1/2})$.

6. Perimeter Sorters: $1 \le M \le N$

We start with the case $N^{1/2} \le M < N$, and then consider $\log N \le M < N^{1/2}$, followed by $M < \log N$.

Case 1: $N^{1/2} \le M < N$. There are two subcases to consider: $T \le (M \log M)^{1/2}$, and $(M \log M)^{1/2} \le T \le (M \log M)^{1/2} \log 2N/M$.

Case 1.1: $T \le (M \log M)^{1/2}$. We use N/M blocks, which each contain an optimal perimeter sorter of M inputs, and which run in time $\Theta(T)$. The outputs of the blocks are combined by an aligned merge. Our merging requires the output from each perimeter sorter to be coded in a form using $M \log 2N/M$ bits (i.e. an encoding that is optimal for representing N numbers in $[0, M-1]$). Appendix D shows how to construct such an encoding from the output of each block. We require that each block have sufficient size to store the encoded data. Thus the area used by a block is $O((M^2 \log M)/T^2 + M \log 2N/M)$. In particular, a block has length $((M \log M)/T)$, which is required to input the M numbers, and width $(M/T + (T \log 2N/M)/\log M)$. The length of all N/M blocks (and hence of the circuit) is therefore $O((N \log M)/T)$. Our encoding will result in the use of $\frac{M \log 2N/M}{T}$ merging structures (and breakpoints), which will thus guarantee that the block width is at least $(T \log 2N/M)/\log M$. There is still one difficulty: when $N/M > T$, it will be impossible to use our bus merging structure to perform an aligned merge of N/M blocks in time $\Theta(T)$. We introduce *terrace trees* (defined below), to replace the bus in our aligned merge. A terrace tree has the following properties:

(i) It is a binary tree.

(ii) The depth from the root to any leaf (block) is at most T (edges).

(iii) Its leaves (blocks) lie on a straight line.

(iv) The physical layout height, h, of the tree is minimal.

[Yao 1981] observed, essentially, that a b-leaf terrace tree must have a layout height of $h = \Omega(\frac{\log b}{\log(2T/\log b)})$, where $T \ge \log b$, and that this bound can be achieved. A construction is given in Appendix F. The tree takes area $O(lh)$, where l is the length of the line on which the leaves lie. Merging (and broadcasting) on the terrace tree proceeds just as merging (and broadcasting) on a merging bus, for locally the bus and the terrace tree have essentially the same structure. We deduce

Lemma 12: For $N^{1/2} \le M \le N$, and $\log N \le T \le (M \log M)^{1/2}$, there exist VLSI perimeter sorters with

$$AT^2 = \Theta\left(NM \log M \log 2N/M \left\lceil \frac{\log 2N/M}{\log\left(\frac{2T}{\log 2N/M}\right)} + 1 \right\rceil\right).$$

Proof: There are N/M blocks, each taking area $O((M^2 \log M)/T^2 + M \log 2N/M)$. We use an aligned merge with $\frac{M \log 2N/M}{T}$ terrace trees, each of layout height (width) $O\left\lceil \frac{\log 2N/M}{\log(2T/\log 2N/M)} + 1 \right\rceil$, and of length $O((N \log M)/T)$. □

Case 1.2: $(M \log M)^{1/2} \le T \le (M \log M)^{1/2} \log 2N/M$. Let $T = x(M \log M)^{1/2}$, where $1 \le x \le \log 2N/M$. We use N/xM blocks, each of which takes x waves of M inputs. Each block

uses an optimal perimeter sorter running in time $\Theta((M\log M)^{1/2})$ to sort sets of M inputs. Each block produces x waves of outputs. Next, we create larger units, called merging structures. Each structure will ultimately merge x waves of outputs from x blocks, and the result will be output in encoded sorted order. Finally, an aligned merge is used to combine the outputs of the structures. As before, the recoding for the aligned merge is done in advance of the actual merge, and each structure needs $O(M\log 2N/M)$ area for this step.

It remains to explain how to build the structures. They are obtained from $\log x$ levels of type 6 merging networks. Recall that the merging networks are configured as a binary tree. A node merges two waves of outputs from its two children. The layout length of a node, the packet length, and the running time all double at successive levels of the tree. The area of a node is proportional to the information content of its one wave of outputs. At the bottom level we choose the type 6 merging networks to have $(M/\log M)^{1/2}$ input wires (and hence width $O((M/\log M)^{1/2})$) and run in time $O((M\log M)^{1/2})$. Because $M<N$ in this instance, the widths (and wire counts) form a decreasing (modified) geometric series, while the running times double at each level of the network. We overlap the running of the various levels of the type 6 merging networks, so that the overall running time is $O(x(M\log M)^{1/2}) = O(T)$.

The total length of the $\frac{N}{Mx^2}$ structures is $\frac{N\log^{1/2}M}{M^{1/2}x} = (N\log M)/T$. Their width, exclusive of the merging network is $O((M\log M)^{1/2})$. After accounting for the aligned merging, we deduce

Lemma 13: For $N^{1/2}\le M\le N$, and $\log N\le T\le \log(2N/M)(M\log M)^{1/2}$, there exist VLSI perimeter sorters

$$\text{with } AT^2 = \Theta\left(MN\log M\log(2N/M)\left\lceil\frac{\log(2N/M)}{\log\left(\frac{2T}{\log(2N/M)}\right)}+1\right\rceil\right).$$

Proof: The aligned merge will use $O((M\log 2N/M)/T)$ terrace trees, each of layout height $O(\frac{\log 2N/M}{\log(2T/\log 2N/M)}+1)$. The length of the network is $O((N\log M)/T)$, and the width of each structure is $O((M\log 2N/M)/T)$. The area of each structure is $\Omega(M\log 2N/M)$, which ensures that the recoders do not consume excessive area. The result follows. □

An AT tradeoff requires the same construction with only $\frac{N\log^{1/2}M}{TM^{1/2}\log 2N/M}$ structures. Each structure is pipelined further to process $\frac{T}{(M\log M)^{1/2}\log 2N/M}$ waves of $M\log^2 N/M$ inputs, in time $O((M\log M)^{1/2}\log 2N/M)$ per wave. (Each wave, in turn, is pipelined in $\log 2N/M$ subwaves, as described above.) Each structure has width $O((M/\log M)^{1/2})$ and length $O((M\log M)^{1/2}\log 2N/M)$. For each structure, after each wave, we merge the outputs of the current wave with the outputs from the previous waves, in time $O((M\log M)^{1/2}\log 2N/M)$, using area $O(M\log 2N/M)$. Following all the waves of inputs, we combine the outputs of the structures using an aligned merge in time $O(T)$. The $\frac{M}{T}\log 2M/N+1$ terrace trees used in the aligned merge each have $O(1)$ height. This yields

Lemma 14: For $N^{1/2}\le M\le N$, and $\log(2N/M)(M\log M)^{1/2}\le T\le \frac{N\log^{1/2}M}{M^{1/2}\log(2N/M)}$, there exist VLSI perimeter sorters with $AT = \Theta(N(M\log M)^{1/2})$.

Case 2: $\log N\le M\le N^{1/2}$. We use $\min[N/(M\log M\log\log M), N/(T(M\log M)^{1/2})]$ blocks, which each take $\max[1, T/(M\log M)^{1/2}\log\log M]$ waves of $M\log M\log\log M$ inputs. Each block sorts a wave of inputs using a perimeter sorter that is described by Lemma 13, and that runs in time proportional to $\min[T, (M\log M)^{1/2}\log\log M]$. Next, the inputs are encoded in a *count*

representation: for each value we take $O(\log M)$ bits to record the number of instances of the value in the inputs. In time $O((M\log M)^{1/2}\log N)$, each block receives at most $M\log M\log N \le M^2\log M$ inputs. In each block, we combine the counts from all the waves of inputs; this combining is carried out following each wave, and takes time $O(\min[T,(M\log M)^{1/2}\log\log M])$, and thus time $O(T)$ over all the waves of input. Blocks have length $l = \Theta(\max[(M\log^2 M\log\log M)/T, \log M(M\log M)^{1/2}])$, and width $\Theta(\max[(M\log\log M)/T,(M/\log M)^{1/2}])$. Thus the area used by a block is $\Omega(M\log M)$, which is sufficient for recoding the inputs in a count representation. We then combine the outputs of $\log N/\log M$ of these blocks. We call the combined $\log N/\log M$ blocks a structure. Since each of the blocks uses area $\Omega(M\log M)$, $\log N/\log M$ blocks use area $\Omega(M\log N)$, and have length $\Theta((l\log N)/\log M)$. Thus there is sufficient area to use $\log N$ bits per count in each structure. The combining of blocks into structures is detailed in the next paragraph. The $\log N$-bit counts output by the structures are summed by $c = \max[1,(M\log N)/T]$ terrace trees. We note that $(l\log N)/\log M \ge c$, so each structure is long enough to allow all c terrace trees to be attached to it. Each terrace tree carries $\min[M,T/\log N]$ counts, each count using $\log N$ bits. It is easy to sum these counts systolically in $O(T)$ time.

The purpose of a structure is to combine the counts for $\log N/\log M \le M/\log M$ blocks. The combining is done using $\max[(M\log M)/T,(M/\log M)^{1/2}]$ channels of width $O(1)$, each of which carries $\min[T/\log M,(M\log M)^{1/2}]$ counts. These counts are summed systolically. Since each structure receives at most M^3 inputs, each count output by the structure will require no more than $3\log M$ bits. The channels run in time $O(T)$. we note that $l \ge \max[(M\log M)/T,(M/\log M)^{1/2}]$, so there is room to attach all the channels to each block. We obtain

Lemma 15: For $\log N \le M \le N^{1/2}$, and $\log N \le T \le (M\log M)^{1/2}\log N$, there exist VLSI perimeter sorters with $AT^2 = \Theta(\frac{MN\log M\log^2 N}{\log(T/\log N)})$.

Proof: We use $\min[N/(M\log M\log\log M), N/(T(M\log M)^{1/2})]$ blocks; thus the sorter has length $O((N\log M)/T)$. Each block has width $O(\max[(M\log\log M)/T,(M/\log M)^{1/2}])$. There are $(M\log N)/T$ terrace trees, each having layout height

$$O(\frac{\log 2N/M}{\log(2T/\log 2N/M)}) = O(\frac{\log N}{\log(2T/\log N)}).$$

Each structure has $\max[(M\log M)/T,(M/\log M)^{1/2}] \le (M\log N)/T$ channels per structure; thus the channels use width $O((M\log N)/T)$ altogether. So the area used is $O(NM\log M\log^2 N/(T^2\log(2T/\log N)))$. □

The same construction gives circuits for larger values of T. Let $T = x(M\log M)^{1/2}\log N$, where $1 \le x \le \log N/\log M$. The input is read in x waves, each wave taking $O((M\log M)^{1/2}\log N)$ time. Our circuit uses the same structures as in Lemma 15 (for time $= \Theta((M\log M)^{1/2}\log N)$) to sort the N/x inputs per wave, but the number of structures (i.e. circuit length) is divided by x. There are $\max[1,M\log N/T]$ terrace trees, so the area is scaled down by a factor of x^2. However, eventually the sorter width fails to decrease as $1/T$. This occurs when the layout width of the terrace trees matches the width of the structures, i.e. when $(M/\log M)^{1/2} = O(\log N/\log(2T/\log N))\cdot((M\log N)/T)$, or when there is only one terrace tree, i.e. when $M\log N/T = 1$. Solving for T gives the requirement $T \le \min[M\log N,(M/\log M)^{1/2}\log^2 N]$. We conclude

Lemma 16: For $\log N \le M \le N^{1/2}$, and $(M\log M)^{1/2}\log N < T \le \min[M\log N,(M/\log M)^{1/2}\log^2 N]$, there exist VLSI perimeter sorters with $AT^2 = \Theta(\frac{MN\log M\log^2 N}{\log(2T/\log N)})$.

For yet larger times, we can, by pipelining further, reduce the number of structures until just one remains, at which point the area is as small as possible. We obtain

Lemma 17: For $\log N \le M \le N^{1/2}$, and $\min[M\log N, (M/\log M)^{1/2}\log^2 N] \le T \le \frac{N(M\log M)^{1/2}}{M\log N}$, there exist VLSI perimeter sorters with $AT = \Theta(N(M\log M)^{1/2} + \frac{N\log N\log M}{\log(2T/\log N)})$.

Case 3, $M < \log N$. Our constructions for very small M are similar but more complex. We start with $\frac{N\log M(M\log M)^{1/2}}{TM^2}$ blocks that take $T/\log M(M\log M)^{1/2}$ waves of M^2 inputs. The blocks are perimeter sorters, as described in lemma 13, that sort each wave of M^2 inputs in time proportional to $\min[T, \log M(M\log M)^{1/2}] = \log M(M\log M)^{1/2}$ and provide the output in count representation. They have length $O(M(M/\log M)^{1/2})$, and width $O((M/\log M)^{1/2})$. We create structures, each of which combines all the waves of output produced by $(\log M \log N)/M$ blocks. We describe how to build the structures in the next paragraph. We complete the sort by combining the outputs of the structures using terrace trees, as above.

A structure is built as follows. The outputs of the blocks are combined using $(M/\log M)^{1/2}$ *column counters*, described in appendix E. A column counter has width $O(1)$. In each structure, the ith counter is pipelined to accumulate, over all waves, the number of instances of the values $i(M\log M)^{1/2}, i(M\log M)^{1/2}+1, \ldots, (i+1)(M\log M)^{1/2}-1$ that occur among the inputs to the structure's blocks, $0 \le i < (M/\log M)^{1/2}$. A counter has one input port per block, and inputs $(M\log M)^{1/2}$ $\log M$-bit counts per wave for each block, one count per value being accumulated. Column counters can input, at one port, one $\log M$-bit number in every $\log M$ time steps. The blocks, during each wave, therefore have sufficient time to transmit their outputs to their connecting counters. The accumulation of data is distributed, so that the total time needed to compute the global counts of each counter's input data equals the time to input all the data plus a delay proportional to the memory capacity of $\Theta(\log M(M\log M)^{1/2} \cdot (\log M \log N)/M)$ bits. Upon accounting for the requisite stage of terrace trees, we obtain

Lemma 18: For $1 \le M \le \log N$, and $\log N \le T \le M\log N$, there exist VLSI perimeter sorters with $AT^2 = \Theta(\frac{MN\log M\log^2 N}{\log(2T/\log N)})$.

Proof: We use $\frac{N\log M(M\log M)^{1/2}}{M^2T}$ blocks; so the length of the sorter is $O((N\log M)/T)$. The width of the blocks is $O((M/\log M)^{1/2})$, and there are $(M/\log M)^{1/2}$ column counters each of width $O(1)$. We use $\max[(M\log N)/T, 1]$ terrace trees, each having layout height $O(\frac{\log N}{\log(2T/\log N)})$. Thus the width of the sorter is

$$O((M/\log M)^{1/2} + \frac{M\log^2 N}{T\log(2T/\log N)} + \frac{\log N}{\log(2T/\log N)}).\square$$

The same proof shows

Lemma 19: For $1 \le M \le \log N$, and $M\log N \le T \le \frac{N(M\log M)^{1/2}}{M\log N}$, there exist VLSI perimeter sorters with $AT = \Theta(N(M\log M)^{1/2} + \frac{N\log N\log M}{\log(2T/\log N)})$.

7. Dense Sorters: $N^2 \le M$

We will be considering sorters that, for large M, run in time much smaller than $\log M$, so we need to specify how the numbers are input: they are input in sets of $2\log N$ contiguous bits. The overall scheme is to sort corresponding pieces of each of the N numbers locally, and to communicate globally as little information as possible, namely the ranks of the inputs. The idea for this parsimonious organization originates in [Leighton 1984]. By introducing some new merging organizations when necessary, and by careful tuning in general, we show how to design such sorting circuits optimally, thereby improving results given in [Leighton

1984], and extending the ranges of area time tradeoffs for M and T. Interestingly, while we obtain an AT^2 tradeoff for sorting, for large enough values of M we do not obtain solely an AT tradeoff; in addition there is an $AT/\log A$ tradeoff (this is due to [Bilardi and Preparata, 1985b]). However, the AT and AT^2 tradeoffs do include all possible times for large enough M for two related problems: ranking, and sorting in the case that the inputs are read twice.

We will use the following sorting circuit as a basic subunit. It inputs the names of N numbers (each expressed in $\log N$ bits), and also inputs the same $2\log N$ consecutive bit variables for each of the numbers, and outputs the name of each number, followed by its local rank. (The rank of a number is the count of numbers it exceeds, and a local rank is the rank of an input based solely on the $2\log N$ input bits for each of the N numbers read into the sorter. Equal numbers, quite clearly, get equal ranks.) A trivial modification of a circuit switching sorter gives such a device. It sorts the N numbers based on their $2\log N$ bits. Next, a binary tree located across the end of the sorter computes, for each number, its rank, which is just the position of the first number having the same value. The $2\log N$ data bits are then replaced by the $\log N$-bit ranks. It is convenient to sort the numbers by name, so a second sorting is performed, using the name as the key. The sorter performs an order preserving compression, taking $2\log N$ bits of the input down to $\log N$ bits. (It also leaves the numbers ordered according to their place of origin.) We call the work done by this device a *compression sort*.

Let $g = \log\left(\frac{\log M}{\log N}\right)$. To sort, when $g\log N \le T \le (N\log N)^{1/2}$, we will use a merging circuit organized as a binary tree of $\log M/\log N$ leaves, where each node contains a compression sorter. Let the leaves be numbered consecutively according to a depth first traversal of the tree. The *ith* leaf receives as input the *ith* set of $2\log N$ consecutive bits belonging to each of the input variables. Each leaf stores its original (unsorted) input data, generates variable names for the data, and applies a compression sort to the input records. The input to each internal node consists of the output from its two children, where for each number (by name) we concatenate the ranks awarded by the children; this concatenation is simple, since the numbers are always output ordered by name.

The true rank of each of the N numbers will be computed at the root of the tree. Once these rankings are computed, it is a simple matter to broadcast them to each of the leaves, whence a final sorting of the original data based on the ranks produces the desired sort.

The tree is laid out as an H-tree, but blocks at different levels are tuned so that they may run at different speeds. The leaves use compression sorters running in time $\Theta(T)$. Each time we go up four levels in the (binary) tree, the running time of the corresponding blocks is halved, until $4\log g$ levels have been traversed. This results in doubling the side length of the blocks as we go up every four levels in the tree. It is now easy to show:

Lemma 20: For $M \ge N^2$, for $\log N \log\left(\frac{\log M}{\log N}\right) \le T \le (N\log N)^{1/2}$, there exist dense VLSI sorters with $AT^2 = \Theta(N^2 \log N \log M)$.

Proof: Each block that is a leaf takes area $O(\frac{(N\log N)^2}{T^2})$, and thus has side length $O(\frac{N\log N}{T})$ (and runs in $O(T)$ time). Since the side length of the blocks only doubles every fourth level in the tree we deduce that the side length of the sorter is $O(l^{1/2}\frac{N\log N}{T})$, where l is the number of leaves in the tree. There are $\frac{\log M}{\log N}$ leaves in the tree, so the area used by the H-tree is $O(N^2 \log N \log M/T^2)$. Since the time to perform the action of a block (essentially sorting) halves every fourth level in the tree, for the first $4\log g$ levels, the time used by these levels is $O(T)$. The remaining levels each run in time $O(T/g)$, and there are $O(g)$ of them;

thus they use a further $O(T)$ time. The result follows. □

We remark that we might expect to build AT^2-optimal sorting circuits using as little as $O(\log\log M)$ time. And in fact, we can obtain optimal results for smaller values of T. We give optimal results for $\log N \cdot \min[\log\log N, \log 2\log\frac{\log M}{\log N}] + \log\log M \le T \le (N\log N)^{1/2}$. By using a slightly different input schedule, one can also obtain results for the remaining smaller values of T (details are left to the interested reader); these results are not known to be optimal.

The key to our constructions is another sorting network, which sorts N $r\log N$-bit numbers, $2 \le r \le N$, in time $O(\log N)$. It is a simple task to modify the mesh of trees (see [Leighton 1984]) to have paths of width r, and to replace the comparators at each grid point by trees of height $\log r$. This will give a circuit taking area $O(N^2r^2\log^2 N)$, that sorts in time $O(\log N)$. We then plug this sorter into the Columnsorter construction of [Leighton 1984]; the only further change is in the circuitry for performing a transpose -- each wire has to be replaced by a bus of width r. This leads to a sorter using area $O(N^2r^2)$ and running in time $O(\log N)$.

Our constructions are similar to the ones above. Again, our sorter can be thought of as a tree of blocks, laid out (essentially) as an H-tree. Let T_b be the running time of the blocks comprising the leaves. Each time we go up four levels in the tree we halve the running time of the corresponding blocks, until the blocks have running time $O(\log N)$. We now rapidly increase the degree of higher level nodes in the tree. Specifically, let the compression sorters at four consecutive levels of the tree input $r\log N$-bit numbers, where $r < N$. Then the next four higher level will use sorters inputing $r^2\log N$-bit numbers. This degree increase stablizes when r reaches N. It is not difficult to see that that the basic H-tree layout organization suffices, since each block has only $N\log N$ bits of output; there is room to provide inputs of $r^2\log N$ bits per number to the blocks at the next level. The change in sorter size at every fourth level ensures that the area used by the higher levels of the tree is negligible.

Lemma 21: For $M \ge N^2$, and $\log N \log 2\log\frac{\log M}{\log N} + \log\log M \le T \le (N\log N)^{1/2}$, there exist dense VLSI sorters with $AT^2 = \Theta(N^2\log N\log M)$.

Proof: Let the leaf blocks run in time T_b. As in the proof of lemma 20, the area used is $O(N^2\log N\log M/T_b^2)$. The time used by the sorters, up to the level of the first sorter running in time $O(\log N)$, is $O(T_b)$. The time used by each of the remaining levels of the tree is $O(\log N)$. There are $O(\min[\log\log N, \log\log\frac{\log M}{\log N}])$ levels to increase r from $r=2$ to $r = \min[N, \frac{\log M}{\log N}]$. There are another $O(\log_N \frac{\log M}{\log N}) = O(\frac{\log\log M}{\log N})$ levels to reach the root of the tree. Thus the total running time is

$$O(T_b + \log N \cdot \min[\log\log N, \log\log\frac{\log M}{\log N}] + \log\log M). \square$$

For $T \ge (N\log N)^{1/2}$, we can obtain AT tradeoffs as usual by pipelining the data through a smaller AT^2 optimal sorter. In Lemma 21, we have constructed trees that run in $\Theta((N\log N)^{1/2})$ time, have as many as $2^{\Theta((N\log N)^{1/2})}$ leaves, and that have an area proportional to the number of input ports. Thus in one $(N\log N)^{1/2}$ timestep we can sort as many as $2^{\Theta((N\log N)^{1/2})}$ contiguous bits of each number. It is easy to see that

Lemma 22: For $\max[\frac{N\log M}{2^{\Theta((N\log N)^{1/2})}}, (N\log N)^{1/2}] \le T \le (N\log N)^{1/2}\frac{\log M}{\log N}$, there exist dense VLSI sorters with $AT = \Theta(N\log M(N\log N)^{1/2})$.

Bilardi and Preparata have shown that there is another area-time tradeoff, and that the tradeoff is tight. For completeness, we state the results and observe that our constructions are

also adequate to establish the existence of such sorting circuits.

Lemma 23: For $2^{\Omega((N\log N)^{1/2})} < A < N\log M$, there exist dense VLSI sorters with $\frac{AT}{\log A} = \Theta(N\log M)$.

Proof: Use a tree of A inputs, as constructed for Lemma 21, where the leaves are sorters running in $\Theta((N\log N)^{1/2})$ time. Then the ranks are computed in $\log A$ time. The tree can be pipelined at that rate for a total time proportional to $T = (\log A)(N\log M)/A$. □

Combining lemmas 22 and 23 gives:

For $\max[\log\log M, (N\log N)^{1/2}] \le T \le (N\log N)^{1/2}\frac{\log M}{\log N}$, there exist dense VLSI sorting circuits with $AT = \Theta(N\log M((N\log N)^{1/2} + \log A))$.

Interestingly, it is possible to extend the Bilardi-Preparata AT lower bound to the problem of computing ranks. In this case, it still turns out that for $A < N\log M$, and $M > N^2$, $AT = \Omega(N\log M(N\log N)^{1/2})$. By pipelining our tree constructions solely at the leaves, we deduce

Lemma 24: For $\max[\log\log M, (N\log N)^{1/2}] \le T \le (N\log N)^{1/2}\frac{\log M}{\log N}$, there exist dense VLSI ranking circuits with $AT = \Theta(N\log M(N\log N)^{1/2})$.

Thus for $T \ge \log N \min[\log\log N, \log\log\frac{\log M}{\log N}] + \log\log M$ these results provide optimal dense VLSI sorting and ranking circuits. Furthermore, if the input data can be read twice (or any finite number of times exceeding one), then the AT tradeoff for ranking circuits becomes a tight tradeoff for sorting (we read the inputs once, compute ranks, and broadcast the ranks to the leaves; then we read the inputs a second time and sort them locally based on their ranks, which gives the desired sort).

8. Perimeter Sorters: $N^2 \le M$

As for the dense sorters with $M \ge N^2$, input bits for each of the N numbers are read in sets of $2\log N$ contiguous bits per block. Again, we obtain an AT^2 tradeoff for sorting. However, there is no AT tradeoff for sorting; the AT tradeoff applies to ranking (and hence sorting, when the inputs can be read twice). Instead, we obtain two other tradeoffs for sorting; they are discussed below. Unfortunately, we are not able to show that most of these bounds are fully optimal (they diverge from the lower bound when M is superpolynomial in N). The fully optimal bounds are the AT tradeoff for ranking, and an AT^2 tradeoff for a sorting verifier. (A sorting verifier inputs N numbers and N ranks in a when- and where-determinate way and indicates whether the rankings are correct, i.e., whether variable X_i has rank r_i, $i = 1,2,\ldots,N$).

Let $g = \log\left(\frac{\log M}{\log N}\right)$. To sort, when $g\log N \le T \le g(N\log N)^{1/2}$, we use a circuit consisting of a binary tree with $\log M/\log N$ leaves. A compression sorter runing in time $O(T/g)$ is located in each node of the tree, and the ranking is computed as in section 7. Thus the rank of each of the N numbers is computed at the root, and broadcast to each of the leaves, whence a final sorting of the data (if available) based on the ranks produces the desired sort.

In fact, the size of the tree can be compressed a little. We use only $\frac{\log M}{g\log N}$ leaves. Each leaf has g phases; in each phase a further $\log N$ bits for each number are read. The current $\log N$ bits (those just read in) are appended to the ranks already computed and new ranks are obtained using a compression sort. The binary tree is laid out as if it were a grounded terrace tree of degree T/g (see appendix F).

A grounded terrace tree has essentially the same structure as a terrace tree except that the processing elements that would belong to each internal node are located in nodes laid out along the same line as the leaves. The internal nodes are just switches that can route data to the processor and back, or can bypass a processor by sending data to the next internal node in the tree. The processors are, essentially, attached to their corresponding switch nodes by edges. Appendix F gives more detail about these structures. A degree d terrace tree is, with one modification, essentially a complete d-ary tree in layout, with each node of a given level at the same height in the layout. The modification replaces each d-ary node by a chain of $d-1$ binary nodes at the same height above the leaves.

The terrace tree has a layout height $\frac{g}{\log(T/g)} + 1$. The value $d = T/g$ is chosen because it is (proportional to) the time in which neighboring P blocks must be able to communicate (see appendix F). Complete binary subtrees of d processors are simulated by a chain of d switches at a given layout height, plus processors, so that the communication time between a processor and its parent (with respect to the subtree) is $O(d)$ in the grounded terrace tree. This time penalty for communication is rendered harmless because the processing time is $\Theta(d)$, so that the overall runing time is at most doubled. The grounding of processors allows the bus structure to have an AT^2 tradeoff (in Lemma 27) even in instances when the number of terrace trees is is less than $(N\log N)^{1/2}$. This is because the height of the sorting structure is the height of the terrace trees multiplied by the number of such trees plus the height of one processor. We deduce

Lemma 25: For $N^2 \leq M$, $A \geq N\log M$, and $g\log N \leq T \leq g(N\log N)^{1/2}$, where $g = \log\left(\frac{\log M}{\log N}\right)$, there exist VLSI perimeter sorters with $AT^2 = O(N^2 \log N \log M\, g\left(\frac{g}{\log(T/g)} + 1\right))$.

Proof: The bound $A \geq N\log M$ is present to ensure we can store all the inputs while the ranks are being computed.

A block takes area $O(g^2N^2\log^2 N/T^2)$, so has side length $O(gN\log N/T)$. As there are $\frac{\log M}{g\log N}$ leaf blocks, the length of the circuit is $O((N\log M)/T)$. The width is the tree layout height times the number of trees (i.e. bus wires) plus the side length of a block, so the width is $O(\frac{gN\log N}{T}\left(\frac{g}{\log(T/g)} + 1\right))$. The result now follows. □

It is worth noting that the area bound can be readily absorbed into the bound for T: $g\log N \leq T \leq (g^{1/2} + g/\log^{1/2} N)(N\log N)^{1/2}$.

As in the dense case, perimeter sorters using area $A < N\log M$ can be also be designed. That is, we pipeline into a sorter that sorts A bits in one phase. We obtain the following results.

Lemma 26: For $N\log N \leq A \leq N\log M$, there exist VLSI sorting circuits with $AT = O(N^{3/2}\log M(\log\frac{A}{N\log N} + \log^{1/2} N \log^{1/2}\frac{A}{N\log N}))$.

Proof: In area A the sorter described in lemma 25 sorts A bits in time proportional to $T' = g'N^{1/2} + (g'N\log N)^{1/2}$, where $g' = \log\frac{A}{N\log N}$. By pipelining the inputs into this sorter and recomputing partial ranks every T' steps, the whole sorter will run in time proportional to $T = \frac{N\log M}{A} \cdot T'$, whence the result follows.

It is also possible to obtain ranking circuits that use less area than the sorting circuits (for some values of T). For $T \leq g(N\log N)^{1/2}$ the construction of lemma 25 suffices without the restriction that $A > N\log M$. Moreover, the construction applies to yet larger values of T;

the only changes are that we pipeline further (at the leaves), and while we cannot slow down the compression sorters (to run slower that $(N\log N)^{1/2}$), we slow down the communication between them. Namely, we increase the packet length to reduce the number of terrace trees (roughly as $1/T$) until the layout height of the tree family is $(N\log N)^{1/2}$. Further pipelining only decreases area in one dimension. It is easy to deduce

Lemma 27: Let $f = (g + \frac{g^2}{\log g + \log N})(N\log N)^{1/2}$. For $N^2 \leq M$, for $g\log N \leq T \leq f$ there exist VLSI ranking circuits with $AT^2 = O(N^2 \log N \log M\, g\left(\frac{g}{\log(T/g)} + 1\right))$, while for $f \leq T \leq (N\log N)^{1/2}\frac{\log M}{\log N}$ there exist VLSI ranking circuits with $AT = \Theta(N\log M\,(N\log N)^{1/2})$.

The limit case arises when we have just one block.

It is interesting to note that we can build verifier circuits for which $AT^2 = \Theta(N^2 \log N \log M\left(\frac{g}{\log(T/g)} + 1\right))$, for $\log N + g \leq T \leq (N\log N)^{1/2}$. In a verifier circuit, a number's rank as well as its value is input, and it outputs a single bit that indicates if the ranks are correct. We use much the same structure, providing the inputs in sets of $\log N$ bits. The binary tree will have $\frac{\log M}{\log N}$ leaves, and each leaf will contain a compression sorter. First, for each number, we broadcast its final rank to every leaf using $(N\log N)/T$ terrace trees. Second, in each leaf, we sort the numbers according to their final rank, in time $O(T)$. Third, for each number we check whether it is larger, equal to, or less than its predecessor, based on the bits available locally. The result takes two bits to store. We lay out the leaves as the leaves of a terrace tree, with path length T. For each node in the tree, for each number, we wish to compute whether the number is larger than, equal to, or smaller than, its predecessor, based on the bits input to the subtree rooted at that node. But for all nodes, except the leaf nodes, this is easily computed in $O(1)$ time per number. For each number this result is calculated by combining the results obtained by the two children of the node. At the root of the tree we are able to check that the input final order is indeed correct, i.e. we check every number is $\geq$ to its predecessor.

The length of the sorter is $O(\frac{N\log N}{T} \cdot \frac{\log M}{\log N}) = O(\frac{N\log M}{T})$. The height of the terrace trees is $O(\frac{g}{\log(T/g)} + 1)$, and there are $((N\log N)/T)$ of them; the width of the blocks is $O((N\log N)/T)$. So the width of the sorter is $O(\frac{N\log N}{T}\left(\frac{g}{\log(T/g)} + 1\right))$. Thus we have $AT^2 = \Theta(N^2 \log N \log M\left(\frac{g}{\log(T/g)} + 1\right))$.

This result is of interest because it suggests that we cannot obtain a better lower bound for ranking large numbers with perimeter sorters, based solely on information flow arguments. For such arguments also apply to the verification problem, but we have given a construction showing the verification lower bound is tight (up to a constant factor). Furthermore, for $T \leq (N\log N + Ng)^{1/2}$, the AT^2 complexities for ranking and sorting are presumably equal, so the verifier results also suggests that information flow arguments are inadequate for improving the lower bound for the sorting problem, in this time range. We also note that the AT^2 complexity for ranking is provably at least as large as the AT^2 complexity for read twice sorting.

9. Conclusions and Remarks

This paper presents a large variety of optimal (and a few nearly optimal) constructions for VLSI sorters, thereby establishing the tightness of several area-time tradeoffs for most relevant number ranges and times. Circuits are given for all M other than the number range $M = N^{1+\Theta(1)}$, which is covered by the earlier works of [Bilardi and Preparata 1984] and [Leighton 1984]. For simplicity, we describe (approximately) the M-T zones for which optimality is still unknown. Dense sorters: $M \approx N$, $\log N \leq T < \log N \log^* N$; and $\log M \approx N$, $\log\log M \leq T < \log N \log\log N$. Perimeter sorters: $\log N = o(\log M)$, most times. We also remark that for the perimeter sorters which contain terrace trees that consume the principal portion of the circuit area, the relevant lower bounds (an application of [Yao, 1981]) presently assume a form of conservative flow in the merging circuits corresponding to the terrace trees; thus these lower bounds are not fully general.

In the process of designing these circuits, we have developed several new merging networks and organizations, of which the aligned merge is, perhaps, the most fundamental. We have also shown new and critical applications for funneled pipelining techniques, and for data compression as well. It turns out that our constructions are typically hybrid, and have three parts, as was discussed in the introduction.

Our constructions give rise to a number of corollaries, including some that are somewhat subtle. These corollaries will be discussed more fully in a later paper; we just state the results here. We consider three functions related to sorting: ranking, packet switching crossbars, and parallel read only memory. Each involves a form of distributed communication.

The ranking problem is to read N input variables, $X_1, X_2, \ldots, X_N$, where $X_i \in [0, M]$, and to output $Y_1, Y_2, \ldots, Y_N$, where Y_i is the rank of X_i. A packet switching crossbar functions as a variant of a packet router, and also as a variant of a scatter-gather operation: the crossbar reads N input indices, $X_1, X_2, \ldots, X_N$, where $X_i \in [0, N-1]$, and N input variables, $W_1, W_2, \ldots, W_N$, where $W_i \in [0, M]$, and outputs $Y_1, Y_2, \ldots, Y_N$, where $Y_i = W_{X_i}$. The parallel read only memory problem is almost the same as the crossbar except that the data W_i are preprocessed (i. e., read in advance, or built into the machine).

We state the results for some of these problems in the ranges of principal interest, and remark that all bounds given below are tight.

Lemma 28: For $N \leq M \leq N^2$, and $\log N \beta(N, M/N) \leq T \leq (N/\log N)^{1/2} \log 2M/N$, There exist VLSI ranking circuits with $AT^2 = N^2 \log^2 \frac{2M}{N}$.

Lemma 29: For $1 \leq M \leq N$, and $\log N \beta(N, M) \leq T \leq (N/\log N)^{1/2} \log 2M$, There exist VLSI N-way packet switching crossbars for $\log M$-bit words with $AT^2 = N^2 \log^2 2M$.

Lemma 30: For $1 \leq M \leq N$, and

$$(N/\log N)^{1/2} \log 2M \leq T \leq \min[(\frac{N}{\log M})^{1/2} \log \frac{\log N}{\log M}, (N \log M)^{1/2}],$$

there exist VLSI N-way packet switching crossbars for $\log M$-bit words with $AT^2 = N^2(\log^2 2M + \log^2 \frac{N \log N}{A})$.

Lemma 31: For $1 \leq M \leq N$, and $\log N \beta(N, M) \leq T \leq (N \log M)^{1/2} \max[\log M, \log \frac{\log N}{\log M}]/\log N$, there exist N-way parallel ROM circuits with $AT = N^{3/2} \log^{1/2} M \max[\log M, \log \frac{A}{N \log M}]$

Lemma 32: For $1 \leq M \leq N$, and

$$(N\log M)^{1/2}\max[\log M, \log\frac{\log N}{\log M}]/\log N \le T \le \log N(N/\log M)^{1/2},$$

there exist N-way parallel ROM circuits with $AT = N^{3/2}\log^{1/2} M\,(\log M + \log\frac{\log N}{\log M})$.

We find it quite surprising these problems should have the same area-time complexity[†] as sorting, for, say $M = N$. Consider, for example, a Parallel Random Access Machine comprising N processors and N bit-memories. With some thought, one can see that a ("smart") optimal sorting network can interconnect the N processors and memories so that for any permutation π stored among the processors, the processors can read in parallel, with processor k reading the bit in memory $\pi(k)$, where $\pi(k)$ is known only to processor k. What is remarkable about this construction is that the memories (provably) cannot "know" which of the processors requested their own data, for the information flow would be too high. Nevertheless, the data can arrive at each of the appropriate processors. Similarly, consider the ranking problem, where N processors are each given one number in $[0,N]$; each processor can learn the rank of its own number despite the fact that the overall communication across any cut line, during the total time T, is only $O(N)$ bits.

[†]There is one difference: we need $N\log N$ area for these problems, rather than $N\log 2N/M$.

10. References

Bilardi G. [1984]. *The Area Time Complexity of Sorting*, Ph.D. Thesis, University of Illinois at Urbana-Champaign.

Bilardi G. and F. Preparata [1984]. "A Minimum Area VLSI Network for $O(\log n)$ Sorting," *Proc. Sixteenth Annual ACM Symposium on the Theory of Computing*, pp. 64–70.

Bilardi G. and F. Preparata [1985a]. "The Influence of Key Length on the Area-Time Complexity of Sorting," *Twelfth International Colloquium on Automata, Languages, and Programming.*

Bilardi G. and F. Preparata [1985b]. "Tessellation Techniques for Area-Time Lower Bounds with Applications to Sorting," CISS, proceedings to appear.

Bilardi G. and F. Preparata [1985c]. "Area-Time Lower Bound Techniques with Applications to Sorting," University of Illinois technical report.

Carter L., R. Floyd, J. Gill, G. Markowsky, M. Wegman [1978]. "Exact and Approximate Membership Testers," *Proc. Tenth Annual ACM Symposium on the Theory of Computing*, pp. 59–65.

Duris P., O. Sykora, C.D. Thompson, I. Vrt'o [1984]. "Tight Chip Area Lower Bounds for Sorting, Selection and Discrete Fourier and Walsh-Hadamard Transformations," manuscript.

El Gammal and K. Pang [1984]. Unpublished manuscript.

Hochschild P.M., E.W. Mayr, A.R. Siegel [1983]. "Techniques for Solving Graph Problems in Parallel," *Proc. Twenty Third Annual Symposium on the Foundations of Computer Science*, pp. 351–359.

Hochschild P.M. [1985]. Thesis, Computer Science Department, Stanford University.

Leighton F.T. [1984]. "Tight Bounds on the Complexity of Parallel Sorting," *Proc. Sixteenth Annual ACM Symposium on the Theory of Computing*, pp. 71–80.

Loui M.C. [1984]. "The Complexity of Sorting on Distributed Systems," *Information and Control*, Jan/Feb/Mar 1984, pp. 70–85.

Siegel A.R. [1984a]. "Optimal Area VLSI Circuits for Sorting," revised text published as "Minimal Storage Sorting Circuits," Special Issue on Sorting, IEEE Transactions on Computers, April, 1985.

Siegel, A.R. [1984b]. "Tight Area Bounds and Provably Good AT^2 Bounds for Sorting Circuits," Technical Report 122, Computer Science Department, NYU, June.

Siegel, A.R. [1984c]. "VLSI Sorters: Techniques for comprehensive AT^2 lower bounds and problems of minimal storage," extended abstract.

Siegel A.R. [1985]. "Computing Information Flow for Lower Bounds in VLSI Circuits," Courant Institute technical report.

Thompson C.D. [1979]. "Area Time Complexity for VLSI," *Proc. Eleventh Annual ACM Symposium on the Theory of Computing*, pp. 81–88.

Thompson C.D. [1980]. *A Complexity Theory for VLSI*, Ph.D. Thesis, Carnegie-Mellon University.

Thompson C.D. [1981]. "The VLSI Complexity of Sorting," *CMU Conference on VLSI Systems and Computations, edited by H.T. Kung, B. Sproull, G. Steele., pp. 108–117.*

Ullman J.D. [1984]. *Computational Aspects of VLSI*, Computer Science Press.

Yao A.C. [1981]. "The Entropic Limitations on VLSI Computations," *Proc. Thirteenth Annual ACM Symposium on the Theory of Computing*, pp. 308–311.

Appendix A. The Encoding

The encodings used to represent a set of X numbers in the range $[0,R]$ are taken principally from [Siegel, 1984a]. To encode $O(X)$ numbers, when $R=X$, $O(X)$ bits are used. $O(R\log 2X/R)$ bits are needed when $R\leq X$, and $O(X\log 2R/X)$ bits when $X^2>R\geq X$. For $R>X^2$, (or, in fact, $R>X^{1+\epsilon}$,) is not possible to use an encoding that decreases the number of bits appreciably.

When $R=X$, we use the following encoding. We use two types of bit-sized markers, black (0) and white (1). A black marker indicates an increment by one, while a white marker indicates an instance of a number; we represent the sorted sequence of numbers by a sequence of these markers. For example: the sequence 0,3,3,4,5,5 of numbers is represented by the sequence 1,0,0,0,1,1,0,1,0,1,1 of markers. Each marker takes one bit; there are X white markers and $X-1$ black markers so $2X-1$ bits are used.

When $X^2>R\geq X$, a black marker indicates an increment by R/X, and uses one bit (0). A white marker represents one number and stores the $\log R/X$ least significant bits of the number. A white marker has a header bit of 1 to distinguish it from a black marker; its length is thus $1+\log R/X$. Again, the numbers are stored in sorted order using these markers. Such a representation takes $O(X\log 2R/X)$ bits.

For $R\leq X$, a black marker indicates an increment by one. It is convenient to use a 0 as a header for such a marker as before, but we also reserve $\log X/R$ additional bits for use as specified below. A white marker (1) is bit-sized, and represents X/R instances of a value, rather than just a single instance as before. Thus the number of white markers between two black markers indicate how many multiples of X/R instances of a specific value appear in the data. The number of additional instances of the value is stored in the $\log X/R$ bits of the black marker following that value. The resulting sequence has bit length $O(R\log 2X/R)$.

When sorting, it will be convenient to store these encodings in several sequences (packets), each of length $O(T)$. For $R=X$, this can be interpreted as dividing the numbers into sets, with each set holding at most T numbers in a range of at most T values. For $R\geq X$, the sets each hold at most $T/\log(2R/X)$ numbers in a range of at most $(RT)/(X\log(2R/X))$ values. And for $R\leq X$, the sets each hold at most $(XT)/R\log(2X/R)$ numbers in a range of at most $T/\log(2X/R)$ values. To encode each of these sets we use $O(T)$ bits, using the appropriate part of the encoding of the whole set, described above. With each of these sets it will often be useful to record further information, such as the value of the smallest number in the set, which is called the *header* of the set. Of course, such auxiliary information will not increase the packet lengths by more than a constant factor.

We also note that the input data need not have a balanced distribution of values, and consequently, some encoded packets might, in fact, be empty, despite having $O(T)$ bits. In addition, several packets might be required to account for all the numbers present in a single packet-range of values. Nonetheless, this encoding modification increases the total encoding length by only a constant factor.

For completeness, we observe that two other data representations are also used. Sometimes it is sufficient to represent values by count, that is, to have R packets of $\log X$ bits, where the binary value in packet k is simply the number of instances of the number k among the inputs. This encoding, is within a constant factor of optimal when, for example, $R<X^{1/2}$. It can also used as a suboptimal encoding in some stages of optimal merging cascades that are outside of any area-time bottleneck.

Finally, to sort very long number strings, i.e. where $R>>X$, the sorting circuits first compute the rank of each number. For this computation, it suffices to represent a substring of each of the the input values by the name of the variable and the rank of the variable's substring, compared to the substrings of identical significance of the other variables. See

sections 7 and 8.

Appendix B. Merging Schemes

This appendix describes the merging schemes used by our dense sorters for $M = N$ and $T = \log N \log^* N$, and describes their construction for this case.

We use a hierarchy of merging networks, indexed by the parameter $x \geq (\log^* N)^3$. A network merges the outputs from an $\frac{x}{\log x (\log^* N)^2} \times \frac{x}{\log x (\log^* N)^2}$ array of units, each of which outputs an encoded sorting of N/x^2 numbers (using $O(\frac{N \log x}{x^2})$ bits). Our network outputs a sorted merge of this data, encoded in a representation for the $\frac{N}{(\log x (\log^* N)^2)^2}$ input numbers, which uses $O(\frac{N \log\log x}{(\log x (\log^* N)^2)^2})$ bits. The network runs in time $O(\log N)$. To accommodate our network, our array must sustain an increase in side length (i.e. the side length exclusive of the side lengths of the $\frac{x}{\log x (\log^* N)^2}$ units) of (at most) $O\left(\frac{N}{\log N \log x (\log^* N)^4}\right)$. We give a description of this merging network (merging network 1) in appendix C.

Suppose we build a nested hierarchy of these merging networks, starting with units for which $x = \log N$. After $\log^* N - \log^* \log^* N$ levels of construction, we arrive at a merging network with $(\log^* N)^3 \leq x \leq (\log^* N)^4$. A unit at the bottom of this sequence uses a circuit switching sorter that runs in time $\log N \log^* N$ and uses area $O(N^2/(\log N \log^* N)^2)$. It also contains an encoding device that uses comparable time and area, whose description can be found in appendix D. Each such unit inputs $N/\log^2 N$ numbers and produces output in encoded form (using $O(N \log\log N/\log^2 N)$ bits). Each level of the merging networks adds side length (less than) $O(N/(\log N (\log^* N)^2))$ to the full sorting network. Altogether, the merging networks add side length $O(N/(\log N \log^* N))$. The initial sort plus the work done by the merging networks takes time $O(\log N \log^* N)$. The sort is not completed, however.

Unfortunately, for $x < \log x (\log^* N)^2$ the merging networks are ineffective, in that they do not cause further merging of the inputs. The underlying problem is that we would be trying to run the merging network too fast; we recall that $\log N \log^* N$ time is needed to send N bits of information across a side of the sorter. So we introduce another hierarchy of merging networks: merging network 2 (described in appendix C). It has the following properties. It holds $const^2$ units arranged in a $const \times const$ array, where $const$ is 16. Let each unit have l output wires. The increase in side length, per row of units, due to the merging network will be $O(l)$.

To ensure that the total area for the hierarchy is small, we increase the packet length at each level of this hierarchy, to prevent the number of packets (i.e. wires) from being excessive. The increased packet length, of course, slows down the merges. Optimality is achieved by balancing this area-time tradeoff.

For simplicity, suppose that the top level of of our type 1 merging network yields a $(\log^* N)^4 \times (\log^* N)^4$ array of units. We use $\log\log^* N$ stages of type 2 merging networks. Stage $\log\log^* N$ outputs the final merge in encoded form; it has $O(N/\log N \log^* N)$ output wires with $O(\log N \log^* N)$ bits per wire. Stage 0 comprises the outputs of our type 1 merging hierarchy; there are $(\log^* N)^8$ units, each with $O(\frac{N}{\log N (\log^* N)^8} \log\log^* N)$ output wires, and $O(\log N)$ bits per wire.

We select a bit-per-wire count, for type 2 merging networks, that forms a geometric sequence. (This choice, of course, fixes the wire-count of a unit, which is proportional to the information content of the unit's input divided by the number of output bits per wire.) Specifically, stage k comprises $\frac{(\log^* N)^8}{256^k}$ units, each with $O(\frac{N}{\log N(\log^* N)^8}128^k(\log\log^* N - k))$ output wires, and $O(2^k \log N)$ bits per wire. Thus the side length used by (the $\frac{(\log^* N)^4}{16^k}$ rows of units in) stage k is $O(\frac{N}{\log N(\log^* N)^4}8^k(\log\log^* N - k))$. The stage runs in time $O(2^k \log N)$. This merging network is described in appendix C.

We can now deduce

Lemma 3: For $M = N$, $T = \log N \log^* N$, there exist VLSI sorters with $AT^2 = \Theta(N^2)$.

Proof: The running time of the type 1 merging networks is $O(\log N \log^* N)$ in total, since each one of them runs in time $O(\log N)$. The running time of the type 2 merging networks is also $O(\log N \log^* N)$, the sum of a simple geometric series. The initial sorters also run in $O(\log N \log^* N)$ time, so the total execution time is just $O(\log N \log^* N)$.

Our sorter has a side length comprising three terms. The length of the initial sorters is proportional to $\log N \times (\frac{N}{\log^2 N \log^* N}) = N/\log N \log^* N$. The additional side length induced by the type 1 merging networks is $O(N/\log N \log^* N)$. The additional side length due to the type 2 merging networks form a (modified) geometric series with total $O(N/\log N \log^* N)$. Hence the total area used is $O(N^2/(\log N \log^* N)^2)$. The result follows. □

Appendix C. Aligned Merging

This merging scheme is used whenever $M < N$. We include it in this appendix for completeness. Its description is the subject of section 2.

Merging Network 1

Let $x \geq (\log^* N)^3$ be a parameter, with $\log N \geq x$. A type 1 merging network for numbers lying in $[0,N]$ is a box containing an array of $\frac{x}{\log x(\log^* N)^2} \times \frac{x}{\log x(\log^* N)^2}$ units, that each output an encoding for N/x^2 sorted inputs (using $O(N \log x/x^2)$ bits). The network merges the outputs of the units, outputing, in encoded form, the sort of its $\frac{N}{(\log x(\log^* N)^2)^2}$ inputs (using $O(\frac{N \log\log x}{(\log x(\log^* N)^2)^2})$ bits). The merging network runs in time $O(\log N)$. The increase in side length of the array due to the presence of the merging network (i.e. the side length exclusive of the side length due to the units) is $O\left(\frac{N}{\log N \log x(\log^* N)^4}\right)$.

The merge proceeds in two phases. First, for each column, we merge the outputs of the $\frac{x}{\log x(\log^* N)^2}$ units in the column; second, we merge the outputs from the first stage. Finally, we recode the outputs into a more compact form. We describe the merge in the first phase; the merge in the second phase is similar. The recoder is described in appendix D.

The outputs from each of the units are produced in packets of $O(\log N)$ bits per wire; each packet is preceded by a $(\log N)$-bit header that equals the value of the smallest number stored in the packet. The encoding satisfies the following *local uniformity criterion*: a packet

encoding sorted numbers from a set of N/x^2 inputs represents at most $\log N/\log x$ numbers in a range of $x^2\log N/\log x$. Thus we use $N\log x/(\log N x^2)$ output wires from each unit in the array. The output packets from each unit in the column are fed into a circuit switching sorter having $N/(x\log N(\log^* N)^2)$ input wires, i.e. one input wire per packet. The packets are sorted by header in time $O(\log N)$. For the moment we assume that for each input value, no unit in the column has more than 2 packets containing the value. (We will soon show how to drop the assumption.) The local uniformity criterion guarantees that each packet of inputs has a range less than $.5\log^3 N$; it can overlap with the values in at most $\log^4 N$ packets. Thus a 'near sort' has been achieved, and a little tidying up will complete the merge.

We now sort groups of $2\log^4 N$ consecutive packets. It suffices to uncode each packet into $O(\log N)$ numbers; we use a circuit switching sorter running in time $O(\log N)$ to sort the numbers, and then reencode them. (Each of these sorters uses area $O(\log^{10} N)$; it is easy to lay them out in the available $N^2/[x\log N(\log^* N)^2]^2$ area.) This process is performed twice; first we sort groups consisting of the packets in positions $2k\log^4 N + i$, $1 \le i \le 2\log^4 N$, $k = 1,2, \cdots$; then we sort groups consisting of the packets in positions $(2k+1)\log^4 N + i$, $1 \le i \le 2\log^4 N$, $k = 1,2, \cdots$. At this point the numbers are in sorted order (in encoded form).

There are many ways to remove the assumption that for each value, no unit has more than 2 packets containing the value. For example, since each level of the merging structure is a perfect sort, it suffices to mark those packets that contain just one value and that have, in the same unit, both preceding and succeeding packets that also contain the value. The unmarked packets are then merged as described above. (The steps needed to partition marked and unmarked packets and to route them to specific input ports of a sorter are based on sorting and can be found in section 2.)

Next, another circuit switching sorter is used to merge by header the sorted unmarked packets with the marked packets. Once sorted, strings of consecutive marked packets will have values in common only with their immediate unmarked predecessor and successor packets. It is a simple matter to use a binary tree to distribute, in $O(\log N)$ time, the unmarked predecessor packet to the appropriate marked packets. Each marked packet will absorb the data that is no less than its header and is less than its successor's header, and the unmarked packet is truncated appropriately. Thus unmarked packets will, in general, be dissected and distributed among the marked packets, which may then acquire new values, and which may double in length. Further sorting operations are used to recode the packets, since some may now be too long, and some may be too short. These recoding steps are addressed in appendix D.

Merging Network 2

Let k be a parameter, with $1 \le k \le \log\log^* N$, and let $j = k-1$. A stage k type 2 merging network for numbers lying in $[0,N]$ is a box containing a 16×16 array of units, each of which has $\dfrac{N}{\log N(\log^* N)^8} 128^j(\log\log^* N - j) = l$ output wires. Each wire carries a packet of length $O(2^j \log N) = b$ bits. The network merges the outputs of the 256 units, putting them into sorted encoded form, in time $O(\log N + b)$. The increase in side length due to the merging network is at most $O(l)$.

We use an 8 phase merge. In each phase we merge the outputs of 2 mergers from the previous phase (in the first phase we merge the outputs of two units). At the end of the merging phases we recode the packets, and redistribute the outputs so that there are $2b$ bits per wire, as described in appendix D. Each phase uses a method similar to that used by merging network 1. The difference is that going from a 'near sort' to a complete sort is simpler. We first sort all the packets by header. Let S_1 and S_2 be the sets of packets we are merging. If a packet P_1 from S_1 (S_2, respec.) has a a packet in S_2 (S_1) as its immediate successor, then P_1 will be truncated and distributed to the string of consecutive packets from S_2 (S_1) that immediately follow it. It suffices to use a binary tree to instantiate the

distribution. The data management details are the same as for a type 1 network. Finally, we note that packets contain $O(b)$ bits, so the sorting and recoding operations take $O(\log N + b)$ time.

Merging Network 3

Let x be a parameter. A type 3 merging network for numbers lying in $[0,N]$ is a box containing a 4×4 array of units, each of which output an encoding for $N/(64x^3)$ sorted inputs. The merging network accumulates 4 waves of outputs from these units, and produces an encoded representation for the sort of these N/x^3 input values. The outputs of each of the 16 units arrive on $N^{1/2}/(8x^{3/2})$ wires; the outputs of the merging network leave on $N^{1/2}/x^{3/2}$ wires. The increase in side length due to the merging network is $O(N^{1/2}/x^{3/2})$. The running time of the network is the maximum of $O(N^{1/2}\log x/x^{3/2})$ and the time for the four waves of inputs to be produced by the units.

The merge has two stages: first, for each of the units, its four waves of output are merged in three merging operations; then, these 16 sets are merged together pairwise as for a type 2 network. Finally, the outputs are distributed over the correct number of output wires. It is easy to see that the network has the required size and running time.

Merging Network 4

A type 4 network merges numbers lying in $[0,M]$, where $M<N$. It is parametrized by x, where $x\geq M$. The network is a box containing an 8×8 array of units, each of which outputs an $O(M\log(2x/M))$-bit encoding representing the sort of x inputs. The merging network produces a $O(M\log(128x/M))$-bit encoding representing the sort of its $64x$ inputs. Let $p = 1/6\log(2x/M)$. The merging network produces its output on $M\cdot 4^{p+1}/T$ wires, in time $O(\max[T\cdot(1/2)^{p+1},\log x])$. The increase in side length due to the merging network is $O(M\cdot 4^{p+1}/T)$. We note that for $x\geq 64M$, the bit length of the output of the merging network is at most twice the bit length of the output of one unit.

The merging is again performed pairwise as for a type 2 network. In this case, however, the recoding is performed by the second type of recoder described in appendix D, since $M<N$.

Merging Network 5

A type 5 network merges numbers lying in $[0,M]$, where $M<N$. It is parametrized by x, where $x\geq(\log^* M)^3$ be a parameter, and $2^{x/(\log^* M)^2}\leq T\log^* M$. The network is a box containing an $(2^{x/(\log^* M)^2}/x)\times(2^{x/(\log^* M)^2}/x)$ array of units, each of which outputs on $\frac{Mx}{T\log^* M}$ wires an $O(M\log x)$-bit encoding representing the sort of Mx^2 inputs. The merging network outputs its $M\,4^{x/(\log^* M)^2}$ inputs in encoded form (using $O(Mx/(\log^* M)^2$ bits). It runs in time $O(T/\log^* M)$, where $\log M\log^* M\leq T\leq\log^2 M$, and has $\frac{M\,2^{x/(\log^* M)^2}}{T\log^* M}$ output wires. The increase in side length due to the merging network is $O\left(\frac{M\,2^{x/(\log^* M)^2}}{T\log^* M}\right)$.

The merging is accomplished in two phases as for the type 1 merging network. We appear to need *aligned merges* for the first merging phase. The outputs from each column of units are combined by an aligned merge, that is, for each column, $Mx2^{x/(\log^* M)^2}$ numbers are merged in an encoded form that uses $O(Mx/(\log^* M)^2)$ bits. Thus there are $\frac{Mx}{T\log^* M}$ channels per column. The increase in side length due to the aligned merging networks for all the columns is thus $\frac{Mx}{T\log^* M}\cdot\frac{2^{x/(\log^* M)^2}}{x} = \frac{M\,2^{x/(\log^* M)^2}}{T\log^* M}$. We remark that there is one difficulty encountered in the aligned merge of section 2 that can be avoided here. The difficulty was

that the set of packets belonging to a single breakpoint range might contain too many items, so that several channels might be requires to merge the numbers in the range. For the current aligned merge, a packet output by a unit will have an encoding that enables it to contain up to $2^{x/(\log^* M)^2} \cdot T\log^* M$ numbers in a range of at most $\frac{T\log^* M}{x}$ values, since that encoding is optimal for the number of inputs comprised by a column. An entire column, therefore, can have at most $\frac{4^{x/(\log^* M)^2}}{x} \cdot T\log^* M$ numbers belonging to a single breakpoint range. Thus it suffices to use an encoding for the units that is optimal for a total of $M\,4^{x/(\log^* M)^2}$ numbers in a range of M values, since this encoding uses only twice as many bits as the encoding we would otherwise have used. Consequently the aligned merge will give a perfect sort rather than just a near sort.

The second phase proceeds in a similar way to the phases in merging network 1. That is, a circuit switched sorter is used to sort the packets by header, in time $\log M + T/\log^* M$. To complete the sort, we mark packets that contain just one value and that have both preceding and succeeding packets that also contain the value. The unmarked packets are then processed. Now an unmarked packet can overlap at most $2\frac{2^{x/(\log^* M)^2}}{x} \cdot \frac{T\log^* M}{x} \le \left(\frac{2T\log^* M}{x}\right)^2 \le 2\log^4 M$ other packets. We sort groups of $4\log^4 M$ contiguous packets, as for merging network 1. To carry out this sort, we expand each marker in each packet into a value and a count, these being the value and count represented by the marker. There are $\le T/\log^* M \le (\log^2 M)/2$ markers per packet, or a total of at most $2\log^6 M$ markers per group, and each count represents at most M^2 numbers. Thus each (expanded) marker uses $O(\log M)$ bits. We sort the markers with a circuit switched sorter that takes $O(\log^{12} M)$ area and runs in time $\log M$. A binary tree is used to count how many numbers of each value are present. The results are then recoded into new sorted packets, in a further $\log M \le T/\log^* M$ time. The marked packets are handled analogously to those for a type 1 merging network.

Merging Network 6

A type 6 merging network is a unit that takes its inputs from two units to its left. One wave of output from this network comprises the merge of two waves of outputs from the two units. Thus in a binary tree of type 6 merging networks, the running time per wave (i.e. the period) doubles at successive levels of the tree. The packet length also doubles, and the wire count is the number of packets needed to carry the encoded data. (The wire count, therefore, roughly doubles or halves, depending on whether $M \ge N$ or not.)

The network accumulates in a buffer its four sets of inputs (as packets), and then sorts and merges them as does a type 2 merging network. The running time is proportional to the length of the output packets. The network uses circuit switching sorters with dimensions proportional to the wire count. Thus the width is proportional to the wire count, while the length is proportional to the packet length, to accommodate the necessary storage.

Appendix D. Recoders

First, we consider the problem of recoding l packets containing altogether P numbers in a range of Q values, $Q \ge P$, where the numbers are currently coded in too sparse a form, that is as if P were smaller than it actually is. So suppose we are given the numbers encoded in b-bit packets on l wires. We show how to recode them using a circuit of size $O(l)$ by $O(l+b)$, in time $O(\log l + b)$. The outputs appear on k wires in packets of b bits each, $k \le l$. (The choice of b implicitly defines k; it is an easy matter to increase or decrease the packet length; just, respectively, reduce or increase the number of packets.)

We introduce the $k/2$ dummy values $i(2Q)/k$, $i = 1,2, \cdots$, and use a circuit switched sorter to merge them with the l packets that are the inputs. We use the header in each packet as its key. If there is a tie between a header and a dummy, the header is defined to be larger. This sort will take time $O(\log l + b)$, on a sorter using area $O(l(l+b))$ (the 'b' term allows for the case that $b \geq l$).

We note that the range of each input packet is at most $2Q/k$ (in fact it is $O(Q/l)$), except in the case that the numbers are not encoded at all (we handle this case below). For each dummy there is just one packet whose values may both precede and follow it: the packet, if any, immediately preceding it in the sorted order just obtained. If there is such a packet we partition it into two packets: those numbers preceding the new value, and the remainder. This remainder is stored with the dummy, which becomes its header. Next, we recode the items in each packet: that is, we introduce more black markers, and for each white marker we reduce the number of bits stored. This can be done in a single sweep over each packet, in time $O(b)$.

We handle the case that the numbers are not initially encoded as follows. As before, we use a circuit switched sorter running in time $O(b)$ to sort the dummies together with the input packets (the key for a packet is the first number in the packet). This sorter uses area $O(l(l+b))$. We then distribute copies of each packet to every dummy with which it overlaps; for each dummy D we remove items from the associated packet so that the remaining items all lie between the value of D and the value of the next dummy. (This can easily be done with the help of a tree of height $\log l$.)

Next, the numbers in each packet are recoded by a sweep across the packet; this entails the introduction of more black markers, and a reduction in the number of bits stored per white marker. It will take time $O(b)$. There are still l, rather than k packets, however. We need to choose new headers: they will be the $k/2$ dummy values we introduced, and every $l/(2k)th$ current header. For each packet, its new header is the nearest preceding header in the sorted order. We copy each newly encoded packet to its new header. There are k new headers, each of which has at most b bits copied to it. This copying takes $O(b)$ time. The problem of routing the new packets to a designated set of k output wires is solved by further sorting as in section 2. Similarly, a tree can be used to find more precise breakpoints.

Second, we consider the problem of recoding P numbers in the range Q, $Q \leq P$, to a denser encoding (the initial encoding supposes there are at least Q numbers present). Suppose the numbers arrive on v wires, in packets of length b, and they are to depart, recoded, on w wires, in packets of length b, where $v \geq w$. We achieve this in time $O(\log v + b)$, using a circuit taking area $O(v(v+b))$.

We introduce dummy headers: every $2Q/wth$ value, and every $2v/wth$ current header. We note that the set of numbers between two adjacent dummy headers can be encoded using b bits. Using a circuit switched sorter, we sort the dummy headers and the packets; the key for a packet is its header. We break ties by defining the dummy headers to be larger. We note that the only packet that may overlap with a dummy header is the one immediately preceding it in the sorted order. We divide such packets into two new packets: the values preceding the dummy header, and the remainder (which take the dummy header as their header). This sorter runs in time $O(\log v + b)$, and uses area $O(v^2)$.

Suppose the current encoding uses counts of at most x bits; we are switching to an encoding allowing counts of up to y bits, $y \geq x$. We start by performing a sweep over each packet to change the counts to the new form, performing permissible additions; this takes time $O(b)$. Then, for those sequences of contiguous packets that hold just one count for the same value, we combine the counts using a tree, in time $O(\log v)$; this tree takes area $O(v \log v)$ (of course, we do not allow the new counts to exceed y bits). Finally, by concatenating the old packets, we form new packets in a further $O(b)$ time.

Our third recoding construction is used in section 6. (Similar recodings are used in section 5; the details are left to the interested reader). The input consists of M numbers in the range 0 to M, stored in $(M/\log M)^{1/2}$ packets, of $(M\log M)^{1/2}$ bits each. We wish to recode these packets to a coding suitable for N numbers in this range ($N \geq M$), where the numbers are to be held in $(M/\log M)^{1/2}\log 2N/M$ packets, each of $(M\log M)^{1/2}$ bits. The available network has size $(M\log M)^{1/2}$ by $(M/\log M)^{1/2}\log 2N/M$. The time available for the recoding is $(M\log M)^{1/2}$. For each of our original set of packets we create $\log 2N/M$ new packets; if the original packet had header h, then the new packets have headers with values $h + i(M\log M)^{1/2}/\log(2N/M)$, for $0 \leq i \leq \log 2N/M - 1$. Each new packet is to take those numbers from the old packet whose values lie in the range between the new packet's header and the next larger new header. This is easily done in $(M\log M)^{1/2}$ time, by first duplicating the original packet $\log 2N/M$ times, and then for each instance of the packet, eliminating all but the appropriate range by a sweep over the packet. Finally, we expand the remaining part of each packet into the new encoding; this takes time proportional to the length of the output, which is at most $(M\log M)^{1/2}$.

Appendix E. Column Counters

Column counters are used to combine the counts of values present among blocks of a perimeter sorter in the case $M \ll N$.

We begin our description of these merging networks by presenting a solution to the following problem. Given N/T input ports lying on a vertical line, count, in time $O(T)$, how many ones are input to all the ports (i.e. add the bits) over time $\Theta(T)$, using a circuit of area $O(N/T)$, where $T^2 \geq N$. We are allowed to separate the input ports as we wish, subject to the condition that the total area (and hence length) is $O(N/T)$.

This problem would be quite simple, were there sufficient width to use a binary tree of adders. The solution is to use an essentially one dimensional pipelined implementation of such a tree. As a consequence of the severe data flow limitations, the one dimensional version is obliged to implement a "poor" binary tree organization where a node always adds bits of a fixed value, and more significant bits are passed up the tree. Then a flushing phase is used to accumulate the data distributed among the tree nodes.

We use a vertical column of $2N/T - 1$ units of width $O(1)$, connected by two shift registers, each of width $O(1)$. One shift register transmits data upward, and one downward. The units are numbered consecutively from 0 to $N/T - 2$. Box number k, where $k = (2r+1)2^{l-1}$ is said to be at level l, so that every second unit is a level 1 unit and every fourth unit is a level 2 unit, etc. Each level 1 unit is connected to an input port. Box k can read and write a bit on both the upward and downward k'th shift register units (the reading and writing occur in adjacent cells of the shift register, the read cell preceding the write cell, with respect to the shift direction). We think of the units as being organized in a tree: that is, the two children of a level i unit, $i > 1$, are the two nearest level $i-1$ units. Each unit knows which way its parent is (up or down), and the root is distinguished. The lengths of the units, however, are not uniform: a level i unit has length $\Theta(i)$. Since there are N/T level 1 units, the total length of the units is $O(N/T)$.

A level i unit counts sets of 4^{i-1} inputs of ones; it has 3 storage bits for this count (i.e. it can count up to 7 sets). Each level i unit has a clock using $O(i)$ bits that counts up to 2^{i+1}. All clocks start at zero at the same time. We remark that it is not necessary to start all the clocks at the same time, although the solution to this problem is well known. It suffices to ensure that their zeros are synchronized. This can be accomplished with an extra shift register, which takes 4 time steps to get from one unit to the next. Initially, a flag bit travels down the register, starting from the top unit (a level 1 unit). A level i unit is passed at time $r \cdot 2^{i+1}$, for some r, which is an appropriate time to set that unit's clock to zero. Non-zero

inputs are allowed to occur only after the bit has traversed the whole shift register.

Each time its clock comes back to zero, a unit must send a count bit to its parent: a zero if it has not accumulated 4 sets of 4^{i-1} ones, and a one otherwise; in the latter case, its local count of sets accumulated is reduced by 4. A level 1 unit takes its input at every time step from its input port. For $i>1$, a level i unit, each time its clock is at $(2^{i-1}-1) \bmod 2^i$, reads its inputs (one from each shift register) and adds these values (each a zero, or a one) to its local set count. At most four sets are counted between a unit's transmissions to its parent, so three bits are sufficient for the local counter. We need to show that no count bit is ever overwritten before it is read, and that the bits are read by the proper units. It suffices to note that bits not yet read pass the output terminal of a level i unit only at times equal to $2^i \bmod 2^{i+1}$, and so never conflict with the bits output by the level i unit (which are output at times equal to $0 \bmod 2^{i+1}$).

At the conclusion of the input phase (time T), the count of the number of ones input will be accumulated in a distributed manner. It is a simple matter to sum the partial counts to obtain a single number. The root can synchronize the leaves (level 1 units) to pass their counts up the tree. It is not difficult to have the counts passed least significant bit first, and to adjust for the missing lesser significant bits among the higher level units. The addition is done systolically. The time for this phase is $O(N/T)=O(T)$.

We now generalize the problem. In each case, the circuit is to have $O(1)$ width.

We are given N/kT input ports lying on a vertical line. Inputs are k-bit binary numbers, occurring at one bit per timestep. We wish to add the values input. The solution is essentially the same: the circuit runs in time $\Theta(T)$, and has length $O(N)$. The shift registers have k bits per unit rather than one, and the units have $k+3$-bit count capacity. Time is k times slower; local clocks have $\log k$ more bits. Addition is ripple (systolic).

Our column counter problem has N/krT input ports lying on a vertical line. Inputs are k-bit binary numbers, occurring at one bit per timestep, but now the k-bit counts at each port are multiplexed as an r vector, so that the bits input at the ports during timesteps $[rks+lk+1, rks+lk+k]$ comprise N/krT k-bit terms for the l'th coordinate. The problem is to sum the $N/(kr)^2$ vectors using area $O(N/T)$, where $T^2 \geq N$. We use essentially the solution given above, except that each storage unit is replicated (vertically) for a total of r stages. Thus each unit has r subunits, each of which keeps a count, but it suffices to use just one clock per unit. The details are straightforward and are omitted.

Appendix F. Terrace Trees

A *terrace tree* of depth T is an N-leaf binary tree subject to the following conditions.

(1) The layout area is minimum (up to a constant factor).

(2) The path length from any leaf to the root is at most T.

(3) The leaves all lie on a straight line.

The parameters h (layout height) and d (degree) will be important in our construction; they satisfy $hd=T$ and $d^h=N$; that is, $h=O(\frac{\log N}{\log(2T/\log N)})$.

We construct an N-leaf d-degree terrace tree as follows. First, a complete N-leaf d-ary tree is laid out, with each node of a given level at the same height in the layout. We now replace each d-ary node by a chain of $d-1$ binary nodes at the same layout height above the leaves; the last node on the chain receives two descending edges from the d-ary node, the other nodes receive one descending edge each, and the first node receives the ascending edge. We note that the path length from any leaf to the root is at most dh, as required.

The second minimum area problem we consider is how to lay out an N-node terrace tree C that emulates a complete binary tree B of N nodes; the requirements for the emulator graph C are formalized as follows.

Let $B = (V, E)$, and $C = (W, F)$. For $v \in V$, let $P(v)$ be the parent of v. Let $m: V \rightarrow W$ be the 1 to 1 map associating nodes of C with those of B. We say that C is a *terrace tree emulator* of B if

(1) C is a terrace tree.

(2) Each leaf of B is mapped by m into a node of C that is a leaf or that has only one child.

(3) For $v \in V$, the length of the path from $m(v)$ to $m(P(v))$ in C is at most d.

(4) If $m(v)$ and $m(P(v))$ are at the same layout height in C, then the interior nodes on the path from $m(v)$ to $m(P(v))$ in C represent (under m) descendants of v in B.

We use the terrace tree described above to construct C. It remains to associate nodes of the binary tree with nodes of the terrace tree. Without loss of generality let d be a power of two (otherwise round down). We associate the top $d-1$ nodes of the binary tree with the nodes at the top layout height (i.e. the top chain of $d-1$ vertices) of the terrace tree. The nodes are arranged in the chain according to an inorder traversal of the $(d-1)$-node subtree of B. Consider the d subtrees of B resulting from deleting the top $d-1$ nodes in B. These subtrees are recursively associated with d subtrees in C, namely the subtrees at the ends of the edges descending from the chain of $d-1$ vertices at the top of C. The subtrees of B are associated with the subtrees of C according to a simultaneous inorder traversal of B and C. It is clear that the path length, in the terrace tree, from any node to its parent (with respect to B) is at most $\frac{d-2}{2} + 2 \leq d$.

A *grounded terrace tree* G is, with a few modifications, a terrace tree emulator of a complete binary tree. The modifications are as follows. The tree G has two kind of nodes, type S and type P. Leaf nodes are type P. Internal nodes are type S. Each type S node w has, in addition to its two original descendents, one other descendent, a type P node located along the line of leaves at the point of intersection with the line that is perpendicular to the leaf-line and that goes through w.